DOI:
10.1016/j.tcs.2018.09.023

Document Version
Peer reviewed version

Conditional Simple Temporal Networks with Uncertainty and Decisions

Matteo Zavatteri[a,*], Luca Viganò[b]

[a] *Dipartimento di Informatica, Università di Verona, Italy*
[b] *Department of Informatics, King's College London, UK*

Abstract

A Conditional Simple Temporal Network with Uncertainty (CSTNU) is a formalism able to model temporal plans subject to both conditional constraints and uncertain durations. The combination of these two characteristics represents the uncontrollable part of the network. That is, before the network starts executing, we do not know completely which time points and constraints will be taken into consideration nor how long the uncertain durations will last. Dynamic controllability (DC) implies the existence of a strategy scheduling the time points of the network in real time depending on how the uncontrollable part behaves. Despite all this, CSTNUs fail to model temporal plans in which a few conditional constraints are under control and may therefore influence (or be influenced by) the uncontrollable part. To bridge this gap, this paper proposes *Conditional Simple Temporal Networks with Uncertainty and Decisions (CSTNUDs)* which introduce *decision time points* into the specification in order to operate on this conditional part under control. We model the dynamic controllability checking (DC-checking) of a CSTNUD as a two-player game in which each player makes his moves in his turn at a specific time instant. We give an encoding into timed game automata for a sound and complete DC-checking. We also synthesize memoryless execution strategies for CSTNUDs proved to be DC and carry out an experimental evaluation with Esse, a tool that we have designed for CSTNUDs to make the approach fully automated.

Keywords: CSTNUD, Dynamic Controllability, Timed Game Automata,

*Corresponding author
Email addresses: matteo.zavatteri@univr.it (Matteo Zavatteri), luca.vigano@kcl.ac.uk (Luca Viganò)

Controller Synthesis, Advanced Temporal Planning Under Uncertainty, ESSE.

---

1. Introduction

1.1. Context and motivations

Temporal networks are a framework to model temporal plans and check the coherence of their temporal constraints which impose a minimal and maximal temporal distance between the occurrence of the events specified in the plan. Temporal plans mainly divide in plans having everything under control and plans having something out of control. The principal components of a temporal network are *time points* and *constraints*. Time points are variables having continuous domain and model the occurrence of events as soon as these variables are assigned real values (i.e., executed). Constraints regulate the minimal and maximal temporal distance between the occurrence of pairs of events and are formalized as linear inequalities.

Whenever both these two components are under control we simply deal with a *consistency* problem asking us to find an assignment of real values to *all* time points satisfying all constraints. *Simple Temporal Networks (STNs)* model exactly this case [1], whereas *Labeled STNs* employed in Drake [2] address temporal plans with choices that are under control; therefore, we keep dealing with a consistency problem asking us to further find suitable values for such choices.

Instead, when some component is out of control, satisfiability is, in general, not enough. In such a case, we deal with a *controllability* problem.

Conditional Simple Temporal Networks (*CSTNs*, [3, 4]) address conditional constraints to enable or disable some parts of the network (i.e., a subset of time points and constraints) during execution. Conditionals are expressed as labels consisting of conjunctions of literals whose atoms are Boolean propositions. The truth value assignments to such propositions are out of control and depend on the behavior of unpredictable external events that are only observed to occur while executing the network.

Simple Temporal Networks with Uncertainty (*STNUs*, [5, 6]) address uncertain (but bounded) durations. Such durations are modeled by contingent links, i.e., pairs of distinct time points specifying a range of allowed values between their distance. One of these time points is called *activation* and it is under control, whereas the other one is called *contingent* and it is not.

The real value assignment to the contingent time points depends again on the behavior of unpredictable external events that are only observed to occur while executing the network.

Conditional Simple Temporal Networks with Uncertainty (*CSTNUs*, [7, 8]) merge the semantics of CSTNs and STNUs addressing conditional constraints and uncertain durations simultaneously.

The controllability of a temporal network implies the existence of a *strategy* operating on the controllable part such that all constraints will eventually be satisfied. Controllability mainly divides in *weak*, *strong* and *dynamic*. Weak controllability ensures the existence of a (possible different) strategy to operate on the controllable part whenever we are able to predict how the entire uncontrollable part will behave before the execution starts. Strong controllability is the opposite case ensuring the existence of a strategy operating always the same way on the controllable part no matter how the uncontrollable part will behave. However, strong controllability is "too strong". If a temporal network is not strongly controllable, it could still be executable by operating on the controllable part reacting to the uncontrollable one as soon as it becomes known. Dynamic controllability addresses exactly this case.

None of the formalisms mentioned so far tackles temporal plans in which some conditional constraints under control may influence (or be influenced by) some uncontrollable part. An initial discussion of this kind of mutual influence is given in [9], where CSTNs are extended with decision nodes regulating the truth value assignments to some propositions under control.

1.2. Contributions

In this paper, we address the modeling and validation of temporal plans in which decisions may influence (or be influenced by) *both* conditional and temporal uncertainty. Our contributions are four-fold:

(1) We provide *Conditional Simple Temporal Networks with Uncertainty and Decisions (CSTNUDs)* as a unified formalism for advanced temporal planning under uncertainty and define dynamic controllability as a two-player game.
(2) We reduce the dynamic controllability checking to the synthesis of a controller for a Timed Game Automaton (TGA) by providing an encoding from CSTNUDs into TGAs.
(3) We automate our approach by providing Esse, a new tool we developed for CSTNUDs. We discuss Esse's input language to specify several kinds

of temporal networks and we compare its performances with those of the previous prototype in [10] by running the same experimental evaluation in which we also consider space consumption and strategy loading time.

(4) We discuss the expressiveness of CSTNUDs by providing a hierarchy of simple temporal networks and showing that (i) all other subformalisms can be encoded into CSTNUDs and (ii) three new formalisms arise implicitly (STNDs, STNUDs and CSTNDs) by augmenting the subformalisms of STNs, STNUs and CSTNs with decisions.

This paper considerably extends and supersedes the work in [10].

1.3. Organization

Section 2 introduces a motivating example we use throughout the paper. Section 3 provides essential background on CSTNUs, TGAs and the DC-checking of CSTNUs via TGAs. Section 4 introduces our main contribution, *CSTNUs with Decisions*, along with a semantics given in *move-based strategies*. Section 5 extends the encoding given in Section 3 to address the DC-checking of CSTNUDs. Section 6 discusses the correctness and complexity of the encoding. Section 7 gives a high level view of the expressiveness of the temporal networks of simple kind providing a hierarchy and showing how all other subformalisms can be encoded into CSTNUDs. It also discusses three new formalisms which arise implicitly. Section 8 discusses Esse, a tool for CSTNUDs along with an experimental evaluation. We compare the performance of Esse with the old prototype (since CSTNUDs were proposed very recently no other tool solving them exists so we at least compare with the preliminary prototype) and we also provide here the algorithm we used to generate random CSTNUD instances. Section 9 discusses related work. Section 10 draws conclusions and discusses future work.

2. Motivating Example

We consider a goods delivery process and in Figure 1 we show its corresponding (temporal) workflow written in the Business Process Modeling Notation BPMN. The workflow starts by processing orders collected online at least 1 hour ago (`Proc0`). This task lasts from 1 to 2 hours and its duration is under control. Then, after exactly 1 hour, the execution flow enters a conditional block and splits into two mutually exclusive branches depending on whether the order requires a one-day delivery or not (diamond labeled by

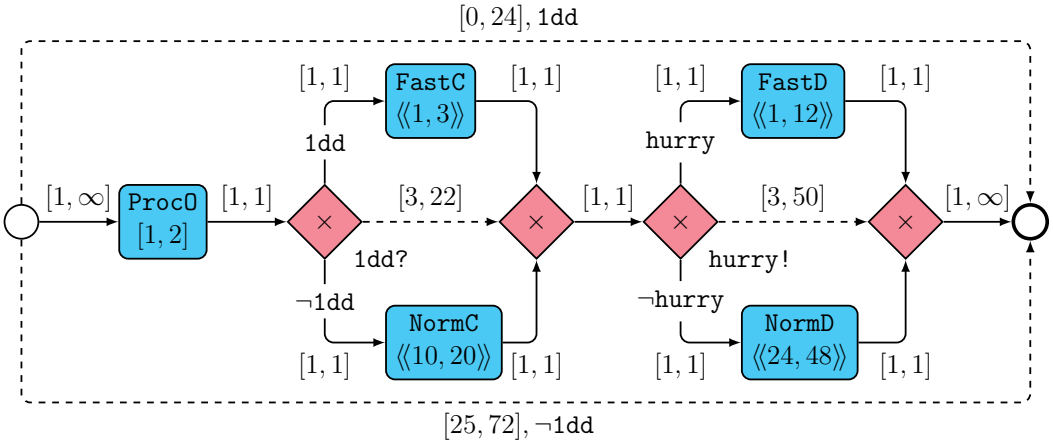


Figure 1: Example of a (structured) temporal workflow in BPMN for a goods delivery process. Diamonds labeled by `conditions?` model uncontrollable choices, whereas those labeled by `decisions!` model controllable ones. Durations represented by $[x, y]$ are controllable, whereas those represented by $\langle\langle x, y \rangle\rangle$ are uncontrollable.

`1dd?`). If `1dd?` $= \top$ (i.e., if `1dd?` is true), then the goods in the order must be delivered within 1 day and a fast collection process (`FastC`) starts after exactly 1 hour since the flow has split. `FastC` lasts from 1 to 3 hours and its exact duration is out of control. Instead, if `1dd?` $= \bot$ (i.e., if `1dd?` is false), then no express delivery is required and the flow of execution continues in the other branch with a normal collection process (`NormC`) starting after exactly 1 hour since the flow has split. `NormC` lasts from 10 to 20 hours and its exact duration is out of control. This conditional block lasts from 3 to 22 hours (dashed edge between the two leftmost diamonds) and concludes after exactly 1 hour since the task in the chosen branch completed.

Now that the shipment is ready, we must choose the correct delivery method in order to satisfy the temporal constraints. Therefore, after exactly 1 hour the flow of execution enters another conditional block but this time we can *decide* which delivery method to use (diamond labeled by `hurry!`). There are two different types of delivery, each one specifying an uncontrollable duration. If we decide to hurry up, then we assign `hurry!` $= \top$ and go for a fast delivery (`FastD`) which takes 1 to 12 hours, whereas if we don't, then we assign `hurry!` $= \bot$ and go for a normal delivery (`NormD`), which takes 24 to 48 hours. The conditional block lasts globally from 3 to 50 hours (dashed edge between the two rightmost diamonds) and completes after one

hour since the task in the chosen branch completed. The whole process completes after minimum another hour. The WF lasts at most 1 day in case of one-day delivery (dashed edge above), and (more than) 1 to 3 days (dashed edge below) otherwise. To guarantee customer satisfaction, the goal of this temporal plan is to always satisfy the temporal constraints no matter what.

3. Background: CSTNUs, TGAs and Dynamic Controllability

In this section we provide essential background on CSTNUs [8], TGAs [11], and the DC-checking of CSTNUs via TGAs [12], which is the only sound and complete approach so far.

3.1. Conditional Simple Temporal Networks with Uncertainty

Given a set $\mathcal{P}$ of Boolean propositions, a *label* $\ell = \lambda_1 \dots \lambda_n$ is any finite conjunction of literals λ_i, where a literal is either a proposition p (positive literal) or its negation $\neg p$ (negative literal). The *empty label* is denoted by $\boxdot$. The *label universe of* $\mathcal{P}$, denoted by $\mathcal{P}^*$, is the set of all possible (consistent) labels drawn from $\mathcal{P}$. For instance, if $\mathcal{P} = \{p, q\}$, then $\mathcal{P}^* = \{\boxdot, p, q, \neg p, \neg q, p \wedge q, p \wedge \neg q, \neg p \wedge q, \neg p \wedge \neg q\}$. Two labels $\ell_1, \ell_2 \in \mathcal{P}^*$ are *consistent* if and only if their conjunction $\ell_1 \wedge \ell_2$ is satisfiable. A label ℓ_1 *entails* a label ℓ_2 (written $\ell_1 \Rightarrow \ell_2$) if and only if all literals in ℓ_2 appear in ℓ_1 too (i.e., if ℓ_1 is more *specific* than ℓ_2). A label ℓ_1 *falsifies* a label ℓ_2 iff $\ell_1 \wedge \ell_2$ is inconsistent. For instance, if $\ell_1 = p \wedge \neg q$ and $\ell_2 = p$, then ℓ_1 and ℓ_2 are consistent since $p \wedge \neg q \wedge p$ is satisfiable, and ℓ_1 entails ℓ_2 since $p \wedge \neg q \Rightarrow p$.

Definition 1 (CSTNU). A *Conditional Simple Temporal Network with Uncertainty* (*CSTNU*) is a tuple $\langle \mathcal{T}, \mathcal{OT}, \mathcal{P}, O, L, \mathcal{L}, \mathcal{C} \rangle$, where:

(1) $\mathcal{T} = \{X, Y, \dots\}$ is a finite set of *time points* (i.e., variables with continuous domain).
(2) $\mathcal{OT} \subseteq \mathcal{T} = \{P?, Q?, \dots\}$ is a set of *observation time points*.
(3) $\mathcal{P} = \{p, q, \dots\}$ is a finite set of Boolean *propositions*.
(4) $O \colon \mathcal{P} \to \mathcal{OT}$ is a bijection associating a unique $P? \in \mathcal{OT}$ to each $p \in \mathcal{P}$ (i.e., $O(p) = P?$).
(5) $L \colon \mathcal{T} \to \mathcal{P}^*$ is a function assigning a *label* to each time point $X \in \mathcal{T}$.
(6) $\mathcal{L}$ is a finite set of *contingent links* (A, x, y, C), where $A, C \in \mathcal{T}$, $0 < x < y < \infty$ $(x, y \in \mathbb{R})$.

(7) $\mathcal{C}$ is a finite set of *labeled constraints* $(Y - X \leq k, \ell)$, where $X, Y \in \mathcal{T}$, $k \in \mathbb{R} \cup \{\pm\infty\}$ and $\ell \in \mathcal{P}^*$. If $(Y - X \leq k, \ell) \notin \mathcal{C}$ for some $\ell \in \mathcal{P}^*$, then $k = \infty$ (for *that* label).

A CSTNU is *well-defined* if and only if all the following properties hold:

(1) For each $X \in \mathcal{T}$, if $\lambda \in L(X)$, where $\lambda \in \{p, \neg p\}$, then $L(X) \Rightarrow L(O(p))$ and $(O(p) - X \leq -\epsilon, L(X)) \in \mathcal{C}$ for some $\epsilon > 0$ (*time point label honesty* [3]).
(2) For each $(A, x, y, C) \in \mathcal{L}$, $A \neq C$ and $L(A) = L(C)$.
(3) For each pair $(A_1, x_1, y_1, C_1), (A_2, x_2, y_2, C_2) \in \mathcal{L}$ if $A_1 \neq A_2$, then $C_1 \neq C_2$.
(4) For each constraint $(Y - X \leq k, \ell) \in \mathcal{C}$, $\ell \Rightarrow L(Y) \wedge L(X)$ (*constraint label coherence* [3]), and for each literal $\lambda \in \ell$, where $\lambda \in \{p, \neg p\}$, $\ell \Rightarrow L(O(p))$ (*constraint label honesty* [3]). □

Label honesty says that whenever a time point X contains a literal p or $\neg p$ in its label $L(X)$, then $L(X)$ implicitly contains $L(P?)$ too because if p has been assigned a value, then $P?$ must have been executed. But for $P?$ to be executed, $L(P?)$ must hold. Therefore, a truth value assignment to p implicitly implies a specific truth value assignment to the propositions in $L(P?)$ (the one which makes $L(P?)$ true). The constraint $(O(p) - X \leq -\epsilon, L(X))$ enforces a positive reaction time saying that we need an ϵ of time to understand what happened and make our next scheduling decision(s). Similar explanations apply for constraint label honesty: a constraint whose label contains p must also contain $L(P?)$.

We execute a time point by assigning it a real value (modeling the occurrence of some temporal event). We execute a CSTNU by executing all relevant non-contingent time points (see below). For any contingent link (A, x, y, C), A is the *activation time point*, whereas C is the *contingent time point*. A is under control, C is not. Once we execute A, we can merely observe the execution of C (by the environment). However, C is guaranteed to occur such that $C - A \in [x, y]$. A contingent link has a unique implicit label given by $\ell = L(A) = L(C)$.

Likewise, an observation time point $P? \in \mathcal{OT}$ is under control, whereas the truth value assignment to its associated Boolean proposition p is not. Once we execute $P?$ we can merely observe such an assignment. Before executing $P?$ the value of p is *unknown*, and after executing $P?$ the value of p is either $\top$ (true) or $\bot$ (false). As we execute observation time points, their

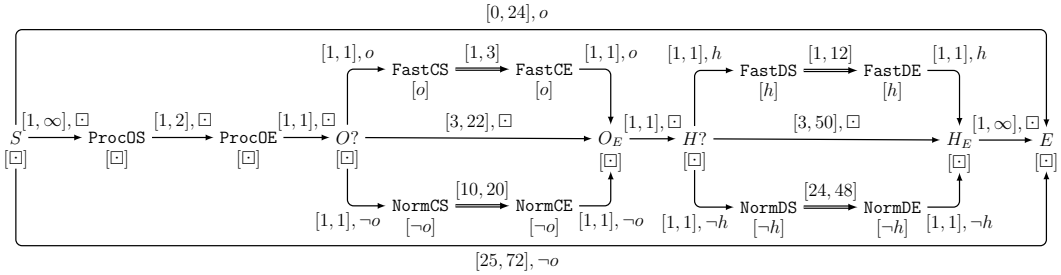


Figure 2: Example of uncontrollable CSTNU modeling the workflow in Figure 1 considering `hurry` as uncontrollable.

truth value assignments to the associated propositions generate the *current partial scenario*. That is, a label $\ell_{cps} \in \mathcal{P}^*$ consisting of the conjunction of these literals. Initially, $\ell_{cps} = \boxdot$, and whenever a proposition is assigned a truth value, the resulting literal λ is appended to ℓ_{cps}. Time points and constraints are *relevant* if their labels are not falsified by ℓ_{cps}. Before executing the network all time points and constraints are relevant. If a time point turns irrelevant, we will not execute it. If a constraint does, we will not be obliged to satisfy it.

A CSTNU is *dynamically controllable* (DC) if there exists a strategy executing all relevant non-contingent time points such that all (relevant) constraints are satisfied no matter which uncertain durations and truth value assignments turn out to be during execution.

We represent a CSTNU graphically as a labeled (multi)graph, where the set of nodes coincides with the set of time points (labels are shown below the nodes), whereas the set of edges divides in contingent links and requirement links. A *contingent link* (shown as a double arrow $A \Rightarrow C$ labeled by $[x, y]$) models $(A, x, y, C) \in \mathcal{L}$. A *requirement link* (shown as a single arrow $X \to Y$ labeled by $[k_1, k_2], \ell$) models the pair $(Y - X \leq k_2, \ell), (X - Y \leq -k_1, \ell) \in \mathcal{C}$.

Consider Figure 1 and assume that `hurry!` is out of control (i.e., that the second conditional split connector is labeled by `hurry?`). Figure 2 is the CSTNU corresponding to Figure 1 according to this modification. Specifically, S and E are two time points modeling the start and the end of the process (i.e., modeling the left and right "circles" in Figure 1). The requirement link `ProcOS` $\to$ `ProcOE` labeled by $[1, 2], \boxdot$ models `ProcO`, where `ProcOS` and `ProcOE` model the start and end of `ProcO`. Likewise, the contingent links $(\texttt{FastCS}, 1, 3, \texttt{FastCE})$, $(\texttt{NormCS}, 10, 20, \texttt{NormCE})$, $(\texttt{FastDS}, 1, 12, \texttt{FastDE})$ and $(\texttt{NormDS}, 24, 48, \texttt{NormDE})$ model `FastC`, `NormC`, `FastD` and `NormD`. The obser-

vation time points $O?$ and $H?$ models the two conditional split connectors, whereas O_E and H_E model their corresponding join connectors. Requirement links $O? \to O_E$ (labeled by $[3, 22]$) and $H? \to H_E$ (labeled by $[3, 50]$) are redundant constraints but they have been intentionally added to help understand "what goes where" in the encoding we discuss in Section 5. If $O?$ assigns $\top$ to o, then `NormCS` and `NormCE` along with the constraints labeled by $\neg o$ turn irrelevant as $\ell_{cps} = o$ falsifies $\neg o$. If $O?$ assigns $\bot$ to o, then we will ignore `FastCS`, `FastCE` and all constraints labeled by o. Likewise, if $H?$ assigns $\top$ (resp., $\bot$) to h, then we will ignore `NormDS` and `NormDE` (resp., `FastDS` and `FastDE`) and all constraints labeled by $\neg o$ (resp., o).

The CSTNU in Figure 2 is uncontrollable. For example, assume that each contingent time point (if relevant) takes its maximal duration. If $\ell_{cps} = o \wedge \neg h$, then the (fastest) execution sequence is $S = 0$, `ProcOS` $= 1$, `ProcOE` $= 2$, $O? = 3$, `FastCS` $= 4$, `FastCE` $= 7$ and $O_E = 8$, $H? = 9$, `NormDS` $= 10$, `NormDE` $= 58$, $H_E = 59$ and $E = 60$ violating $[0, 24], o$ between S and E requiring that E is executed within 24 when $o = \top$.

Also, if $\ell_{cps} = \neg o \wedge \neg h$, then the (fastest) execution sequence is $S = 0$, `ProcOS` $= 1$, `ProcOE` $= 2$, $O? = 3$, `NormCS` $= 4$, `NormCE` $= 24$ and $O_E = 25$, $H? = 26$, `NormDS` $= 27$, `NormDE` $= 75$, $H_E = 76$ and $E = 75$ violating $[25, 72]$ between S and E requiring that E is executed after minimum 25 and within 72 since S occurred).

3.2. Timed Game Automata

A *Timed Automaton* (TA) [13] refines a Finite Automaton [14] by adding real-valued *clocks* and *clock constraints*. All clocks increase at the uniform rate and may be reset many times.

Definition 2 (TGA). A *Timed Automaton (TA)* is a tuple $\langle Loc, Act, \mathcal{X}, \to, Inv \rangle$, where

(1) Loc is a finite set of *locations* (including an initial location). A location is *urgent* if time freezes in it.
(2) Act is a finite set of *actions*.
(3) $\mathcal{X}$ is a finite set of *real-valued clocks*.
(4) $\to \subseteq Loc \times \mathcal{H}(\mathcal{X}) \times Act \times 2^{\mathcal{X}} \times Loc$ is the transition relation. An edge (L_i, G, A, R, L_j) represents a transition from L_i to L_j realizing action A. $G \in \mathcal{H}(\mathcal{X})$ is a *guard* consisting of a conjunction of clock constraints having the form $c_1 \sim k$ or $c_1 - c_2 \sim k$ where $c_1, c_2 \in \mathcal{X}$, $k \in \mathbb{N}$ and

$\sim \in \{<, \leq, =, >, \geq\}$. $R \subseteq 2^{\mathcal{X}}$ is the set of clocks to reset (i.e., the set of clocks to set to 0).

(5) $Inv : Loc \to \mathcal{H}(\mathcal{X})$ is a function assigning an *invariant* (modeled as a conjunction of clock constraints) to each location. $Inv(L)$ says *when* the TA is allowed to remain in L.

A *Timed Game Automaton (TGA)* [11] extends a TA by dividing transitions into *controllable* and *uncontrollable*. Uncontrollable transitions have priority over controllable ones. □

We represent a TGA graphically as a (multi)graph where the set of nodes coincides with that of locations, whereas the set of edges models controllable transitions (solid edges) and uncontrollable ones (dashed edges). Figure 3 depicts a TGA encoding the CSTNU in Figure 2. In what follows, we sum up how this encoding is achieved and dynamic controllability is checked.

3.3. Dynamic Controllability of CSTNUs via TGAs

The *DC-checking problem* is the problem of deciding if a CSTNU is DC. We can answer the DC-checking problem by using *sound* and *complete* TGA reachability algorithms [12, 15]. We model the DC-checking as a two-player game between a controller (`ctrl`) and the environment (`env`). The goal of `ctrl` is to reach a specific location as soon as all relevant time points have been executed and all constraints are satisfied, whereas the goal of `env` is to prevent `ctrl` from doing that. If `ctrl` wins, the network is DC, otherwise it is not. An important aspect of this encoding is that `ctrl` is assigned *uncontrollable* transitions, whereas `env` is assigned *controllable* ones. This is necessary to allow `env`'s instantaneous reactions as in the TGA semantics uncontrollable transitions go first [12, 15, 16]. The encoding is as follows.

Clocks. $\mathcal{X}$ contains a clock $\mathtt{cX}$ for each time point $X \in \mathcal{T}$ and a clock $\mathtt{bP}$ for each proposition $p \in \mathcal{P}$. $\mathcal{X}$ also contains two special clocks $\hat{\mathtt{c}}$ (modeling the global time) and $\mathtt{c}_\delta$ (regulating the interplay of the game). $\mathtt{cX} = \hat{\mathtt{c}}$, means that X has not been executed, whereas $\mathtt{cX} < \hat{\mathtt{c}}$ means that X was executed at time $\hat{\mathtt{c}} - \mathtt{cX}$ (when this difference is > 0). Likewise, $\mathtt{bP} = \hat{\mathtt{c}}$ means that $p = \top$, whereas $\mathtt{bP} < \hat{\mathtt{c}}$ means that $p = \bot$ (both when $\mathtt{cP} < \hat{\mathtt{c}}$). Each `cX` and `bP` may be reset at most once. For our example we have $\mathcal{X} = \{\hat{\mathtt{c}}, \mathtt{c}_\delta, \mathtt{cS}, \mathtt{cPOS}, \mathtt{cPOE}, \mathtt{cO}, \mathtt{cFCS}, \mathtt{cFCE}, \mathtt{cNCS}, \mathtt{cNCE}, \mathtt{cOE}, \mathtt{cH}, \mathtt{cFDS}, \mathtt{cFDE}, \mathtt{cNDS}, \mathtt{cNDE}, \mathtt{cHE}, \mathtt{cE}, \mathtt{bO}, \mathtt{bH}\}$.

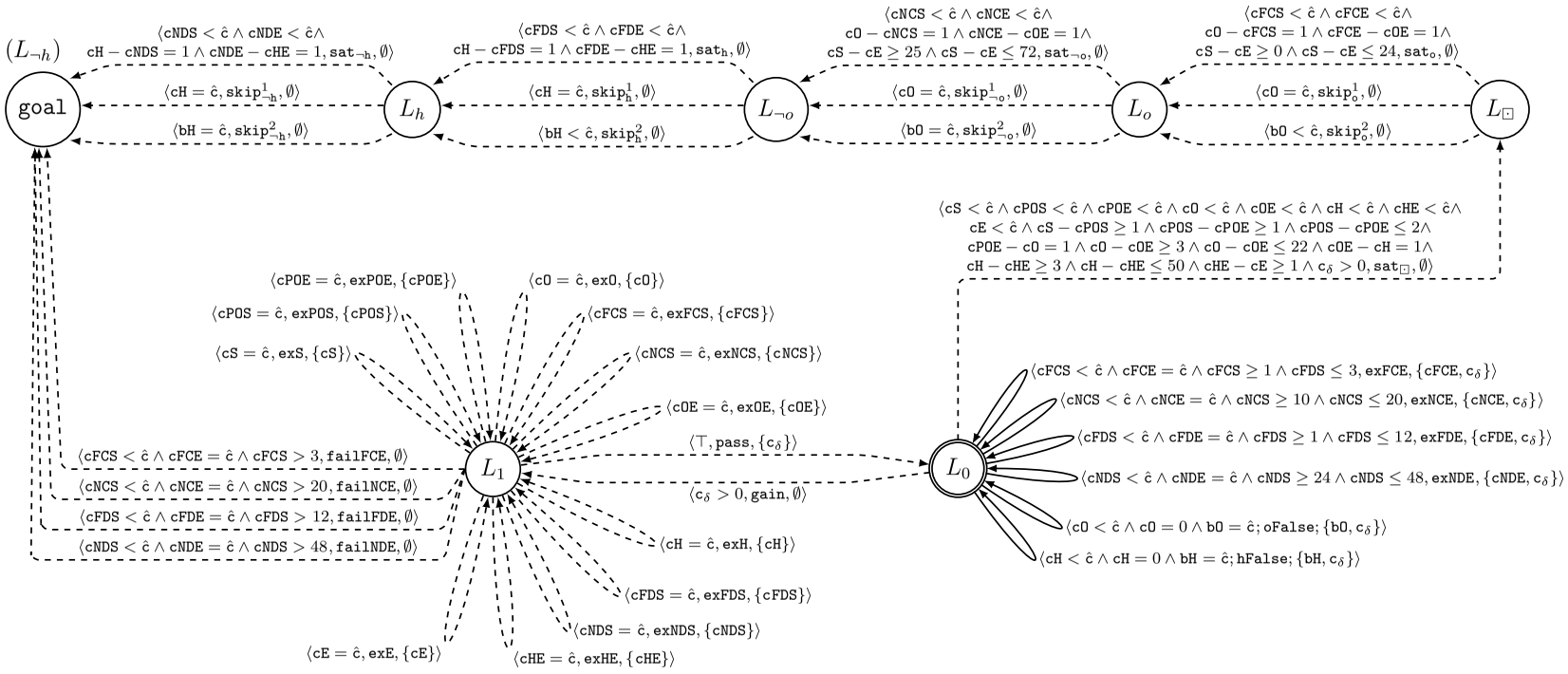


Figure 3: TGA encoding the CSTNU in Figure 2. L_0 is the initial location, L_1, $L_{\boxdot}$, L_o, $L_{\neg o}$, L_h and goal ($L_{\neg h}$) are urgent. Solid (resp., dashed) edges model controllable (resp., uncontrollable) transitions.

Locations. *Loc* contains three core locations L_0 (initial), L_1 (urgent) and $\texttt{goal}$ (urgent), and $n-1$ urgent locations $L_{\ell_1}, \ldots, L_{\ell_{n-1}}$, where n is the number of distinct labels in the CSTNU. That is, $n = |\{L(X) \mid X \in \mathcal{T}\} \cup \{\ell \mid (Y - X \leq k, \ell) \in \mathcal{C}\}|$. For our example, $Loc = \{L_0, L_1, L_{\boxdot}, L_o, L_{\neg o}, L_h, \texttt{goal}(= L_{\neg h}\}$ as the distinct labels are $\{\boxdot, o, \neg o, h, \neg h\}$.

Transitions. $\rightarrow$ contains controllable and uncontrollable transitions to model the following:

- *Game interplay.* `pass` and `gain` are uncontrollable transitions regulating the game interplay. In particular `gain` can be taken only when $\texttt{c}_\delta > 0$ modeling the *reaction time* needed to observe how the uncontrollable part behaves.
- *Non-contingent time point executions.* For each non-contingent time point X there is an uncontrollable self-loop transition $\langle L_1, \texttt{cX} = \hat{\texttt{c}}, \texttt{exX}, \{\texttt{cX}\}, L_1\rangle$ modeling the execution of X. The guard says that X has not been executed yet, while the reset fixes the execution time of X to $\hat{\texttt{c}} - \texttt{cX}$ by resetting $\texttt{cX}$.
- *Contingent time point executions.* For each contingent link $(A, x, y, C) \in \mathcal{L}$ there is a controllable self-loop transition $\langle L_0, \texttt{cA} < \hat{\texttt{c}} \wedge \texttt{cC} = \hat{\texttt{c}} \wedge \texttt{cA} \geq x \wedge \texttt{cA} \leq y, \texttt{exC}, \{\texttt{cC}, \texttt{c}_\delta\}, L_0\rangle$ to allow `env` to execute the contingent time point C such that $C - A \in [x, y]$, and a fail transition $\langle L_0, \texttt{cA} < \hat{\texttt{c}} \wedge \texttt{cC} = \hat{\texttt{c}} \wedge \texttt{cA} > y, \texttt{failC}, \{\emptyset\}, \texttt{goal}\rangle$ to allow `ctrl` to move to `goal` if `env` fails or refuses to take the transition.
- *Truth value assignments.* For each proposition $p \in \mathcal{P}$ there is a controllable self-loop transition $\langle L_0, \texttt{cP} < \hat{\texttt{c}} \wedge \texttt{cP} = 0 \wedge \texttt{bP} = \hat{\texttt{c}}, \texttt{pFalse}, \{\texttt{bP}, \texttt{c}_\delta\}, L_0\rangle$ to allow `env` to assign $\bot$ to p, if it decides so. If it does not, the truth value of p will remain forever $\top$.
- *Winning conditions.* To check that all relevant time points have been executed and all constraints are satisfied we connect each pair of locations $(L_{\ell_{i-1}}, L_{\ell_i})$ in the winning path $L_0 \rightarrow L_{\boxdot} \rightarrow \cdots \rightarrow L_{\ell_{n-1}} \rightarrow \texttt{goal}$ by means of a set of uncontrollable transitions. Each set of transitions going from $L_{\ell_{i-1}}$ to L_{ℓ_i} verifies that if ℓ_{cps} satisfies ℓ_i, then all time points labeled by ℓ_i must have been executed and all constraints labeled by ℓ_i are satisfied. If ℓ_{cps}

falsifies ℓ_i, a `skip` transition allows us to ignore this check. In this way, the problem is decomposed with respect to the specific labels avoiding the combinatorial explosion of all arising cases. For example, the set of transitions going from $L_{\boxdot}$ to L_o is generated as follows. In the scenario where $O?$ has been executed and o assigned $\top$ (i.e., $\ell_{cps} = o$), then `FastCS` and `FastCE` must have been executed, and $\texttt{FastCS} - O? \leq 1$, $O? - \texttt{FastCS} \leq -1$, $O_E - \texttt{FastCE} \leq 1$, $\texttt{FastCE} - O_E \leq -1$, $E - S \leq 24$ and $S - E \leq 0$ are satisfied. In other words, the meta conditional constraint $(\texttt{cO} < \hat{\texttt{c}} \wedge \texttt{bO} = \hat{\texttt{c}}) \implies (\texttt{cFCS} < \hat{\texttt{c}} \wedge \texttt{cFCE} < \hat{\texttt{c}} \wedge \texttt{cO} - \texttt{cFCS} = 1 \wedge \texttt{cFCE} - \texttt{cOE} = 1 \wedge \texttt{cS} - \texttt{cE} \geq 0 \wedge \texttt{cS} - \texttt{cE} \leq 24)$ refines to $\neg(\texttt{cO} < \hat{\texttt{c}} \wedge \texttt{bO} = \hat{\texttt{c}}) \vee (\texttt{cFCS} < \hat{\texttt{c}} \wedge \texttt{cFCE} < \hat{\texttt{c}} \wedge \texttt{cO} - \texttt{cFCS} = 1 \wedge \texttt{cFCE} - \texttt{cOE} = 1 \wedge \texttt{cS} - \texttt{cE} \geq 0 \wedge \texttt{cS} - \texttt{cE} \leq 24)$ simplifying to $(\texttt{cO} = \hat{\texttt{c}}) \vee (\texttt{bO} < \hat{\texttt{c}}) \vee (\texttt{cFCS} < \hat{\texttt{c}} \wedge \texttt{cFCE} < \hat{\texttt{c}} \wedge \texttt{cO} - \texttt{cFCS} = 1 \wedge \texttt{cFCE} - \texttt{cOE} = 1 \wedge \texttt{cS} - \texttt{cE} \geq 0 \wedge \texttt{cS} - \texttt{cE} \leq 24)$ since TGAs do not allow negations nor disjunctions of clock constraints in the guards. Finally, we generate a transition[1] for each disjunct $(\texttt{sat}_\texttt{o}, \texttt{skip}^1_\texttt{o}, \texttt{skip}^2_\texttt{o})$.

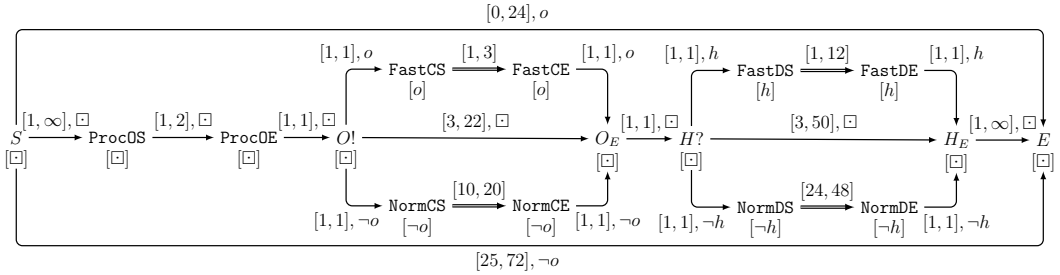
DC-checking is reduced to looking for a control strategy for `env` to always prevent `ctrl` from getting to `goal`. If such a strategy exists, the initial CSTNU is not DC, otherwise it is (as `ctrl` has a counter-strategy to react to any combination of `env`'s moves).

4. CSTNUs with Decisions

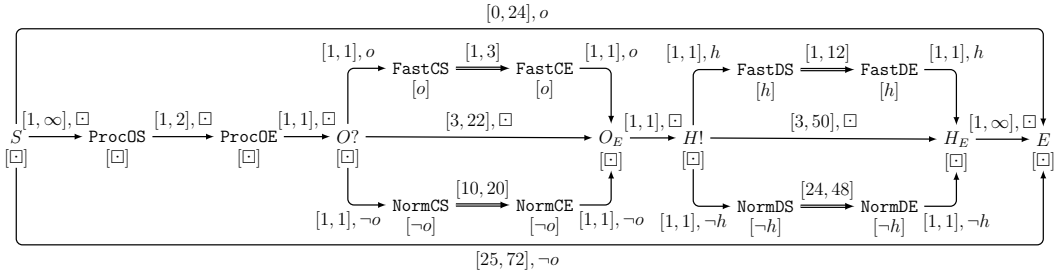
4.1. Syntax

In this section, we extend CSTNUs by injecting a new kind of time point: the *decision time point*. A decision time point $D!$ dualizes an observation time point $P?$ as the truth value assignment to the associated proposition is *under control*. As a result, the controllable and uncontrollable parts may now mutually influence one another. That is, deciding some truth value may restrict (or even exclude) some uncontrollable part and vice versa. Several interesting cases may arise depending on whether a few truth values are

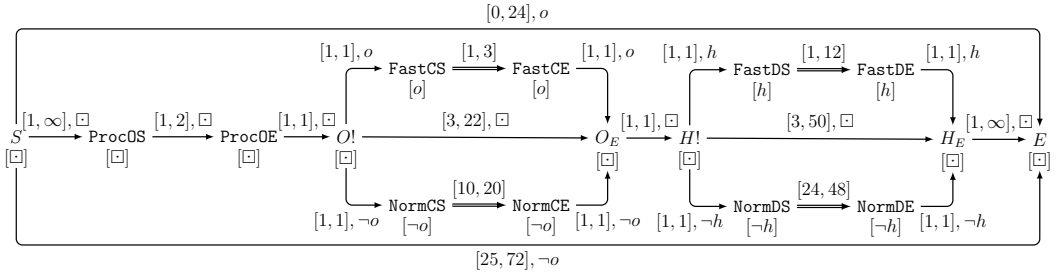
[1] We model $Y - X \leq k$ as $(\hat{\texttt{c}} - \texttt{cY}) - (\hat{\texttt{c}} - \texttt{cX}) \leq k$ simplifying to $\texttt{cX} - \texttt{cY} \leq k$. We write $\texttt{cX} - \texttt{cY} \geq k$ to abbreviate $X - Y \leq -k$ and $\texttt{cX} - \texttt{cY} = k$ to abbreviate the pair $Y - X \leq k$ and $X - Y \leq -k$.



(a) *A decision before any uncontrollable part.*



(b) *A decision after all observations and some uncontrollable durations.*



(c) *A decision after another decision and some uncontrollable durations.*

Figure 4: Possible cases of the CSTNU in Figure 2 when substituting decision time points for observation time points.

decided before or after having full information on how the uncontrollable part will behave or has behaved. Consider, for example, Figure 4 in which we took the initial CSTNU in Figure 2 and substituted decision time points for observation ones in all possible combinations. We discuss Figure 4a-4b-4c focusing on the combinations of minimal and maximal durations of contingent links only. If it works for them, then it must work for any other combination of durations.

(1) In Figure 4a $O!$ is a decision time point. The resulting CSTNUD is uncontrollable. If we *decide* o (i.e., assign $\top$ to o), then *observe* $\neg h$ (i.e., $H?$ assigns $\bot$ to h) and `FastCE`, `NormDE` take their maximal durations, and then the fastest possible execution will execute E at 60 violating $(E - S \leq 24, o)$ as S is executed at 0. Conversely, if we decide $\neg o$, then observe $\neg h$ and `NormCE` and `NormDE` take their maximal durations, then we will have to execute E at 78 (violating $E - S \leq 72, d$).

(2) In Figure 4b $H!$ is a decision time point. The resulting CSTNUD is DC. We always execute S at 0, `ProcOS` at 1, `ProcOE` at 2 and $O?$ at 3. Then, assume that we observe o. Regardless of the duration of `FastCE`, we can only decide h. Indeed, if we decided $\neg h$, then, regardless of the duration of `NormDE`, we would have to execute E always after time 24, violating $(E - S \leq 24, o)$. Assume now that we observe $\neg o$. If `NormCE` takes its minimal duration, $\neg h$ is the only good decision. If we decided h and then `FastDE` took its minimal duration, we would execute E at 20, violating $(S - E \leq -25, \neg o)$. On the contrary, if `NormCE` takes its maximal duration, then we can only decide h. If we decided $\neg h$ and `NormDE` took its maximal duration, then we would have to execute E at 77, violating $(E - S \leq 72, \neg o)$.

(3) In Figure 4c $O!$ and $H!$ are both decision time points. The resulting STNUD (i.e., a CSTNUD without the "C" meaning no observation time points) is of course dynamically controllable.[2] If we decide o, then deciding h is always going to be fine. If we decide $\neg o$, then we will decide either h or $\neg h$ depending on how long `NormCE` lasts. If `NormCE` takes its minimal duration, then we will decide $\neg h$ (but not h since `FastDE` could then take its minimal duration). If `NormCE` takes its maximal duration, then we will decide h (but not $\neg h$ since if `NormDE` could then take its maximal duration).

Hence, *decisions are dynamic*.

Definition 3 (CSTNUD). A *Conditional Simple Temporal Network with Uncertainty and Decisions* (*CSTNUD*) is a tuple $\langle \mathcal{T}, \mathcal{OT}, \mathcal{DT}, \mathcal{P}, O, L, \mathcal{L}, \mathcal{C} \rangle$, where:

[2] If a network is DC (e.g., Figure 4b), then turning controllable some uncontrollable part (e.g., Figure 4c) *cannot worsen* the situation as it can't make the network uncontrollable (it is like turning a $\forall$ into an $\exists$). The vice versa does not hold in general.

(1) $\mathcal{T}, \mathcal{OT}, \mathcal{P}, L, \mathcal{L}, \mathcal{C}$ are exactly the same as those given for CSTNUs. We denote the set of contingent time points as $\mathit{Contingent} = \{C \mid (A, x, y, C) \in \mathcal{L}\}$.

(2) $\mathcal{DT} \subseteq \mathcal{T} = \{D!, E!, \dots\}$ is a set of *decision time points* such that $\mathcal{OT} \cap \mathcal{DT} = \emptyset$.

(3) $O \colon \mathcal{P} \to \mathcal{DT} \cup \mathcal{OT}$ is a bijection associating a unique observation or decision time point to each proposition. If $O(p) \in \mathcal{OT}$, then p is called *observable*, whereas if $O(d) \in \mathcal{DT}$, then d is called *decidable*. $\mathcal{OP} = \{p \mid O(p) \in \mathcal{OT}\} \subseteq \mathcal{P}$ and $\mathcal{DP} = \{d \mid O(d) \in \mathcal{DT}\} \subseteq \mathcal{P}$ are the sets of all observable and decidable propositions, where $\mathcal{OP} \cap \mathcal{DP} = \emptyset$ (any proposition is either under control or out of control).

A CSTNUD is *well-defined* iff the underlying CSTNU is well-defined and time point label honesty extends to decidable propositions as follows: For each $X \in \mathcal{T}$, if $\lambda \in L(X)$, where $\lambda = \{d, \neg d\}$ and $d \in \mathcal{DP}$, then $L(X) \Rightarrow L(O(d))$ and $(O(d) - X \leq 0, L(X)) \in \mathcal{C})$. That is, X can be executed at the same time of $D!$ (but instantaneously after $D!$ since time points executed at the same instant must follow an *order of execution*). Moreover, if $D! \in \mathcal{DT}$, where $D! = O(d)$, then we have that $d \notin L(D!)$ (no auto labeling). □

The last sentence of Definition 3 serves to avoid the following circularity: If $d \in L(D!)$, then $D!$ can be executed if and only if $d = \top$ (i.e., instantaneously after itself), but d can be assigned a truth value only upon the execution of $D!$ (deadlock). This problem does not happen with observation time points as time point label honesty would insert the implicit negative loop $(O(p) - P? \leq -\epsilon)$. However, $(O(d) - D! \leq 0)$ trivially holds for decision time points, thus we must exclude auto labeling on decision time points.

4.2. Execution semantics

We give the execution semantics of a CSTNUD as a two-player game in which `Player1` models the controller and `Player2` models the environment. We employ *execution sequences* [17] to model the state of the game and define players' strategies as mappings from execution sequences considered at specific time instants to *moves*.

A *sequence* $\langle x_1, x_2, \dots, x_n \rangle$ is a totally ordered collection of elements such that for any pair of elements x_i, x_j, if $i < j$ (resp., $i > j$), then it means that x_i is before (resp., after) x_j. We slightly abuse notation and write $\langle x_1, x_2, \dots, x_n \rangle \cup \langle x_p \rangle$ to mean the append operation resulting in

$\langle x_1, x_2, \ldots, x_n, x_p \rangle$, where $n < p$. We write $x_i \in \langle x_1, x_2, \ldots, x_n \rangle$ iff there exists $j \in \mathbb{N}$, $1 \leq j \leq n$ such that $x_i = x_j$ (membership), and $|\langle x_1, x_2, \ldots, x_n \rangle| = n$ (cardinality).

A *partial schedule* for a subset of time points $\mathcal{T}' \subseteq \mathcal{T}$ is a mapping $S_{\mathcal{T}'} \colon \mathcal{T}' \to \mathbb{R}$ assigning a real value to each $X \in \mathcal{T}'$. A *partial schedule* for a subset of Boolean propositions $\mathcal{P}' \subseteq \mathcal{P}$ is a mapping $S_{\mathcal{P}'} \colon \mathcal{P}' \to \{\top, \bot\}$ assigning either $\top$ or $\bot$ to each $p \in \mathcal{P}'$. We write $\mathtt{b}$ for a generic Boolean value (i.e., $\mathtt{b} \in \{\top, \bot\}$). We write $S_{\mathcal{T}'} \cup \{S_{\mathcal{T}'}(Y) = k\}$ for the extension of the domain of $S_{\mathcal{T}'}$ by adding a time point Y such that $S_{\mathcal{T}'}(Y) = k$. Similarly, we write $S_{\mathcal{P}'} \cup \{S_{\mathcal{P}'}(p) = \mathtt{b}\}$ for Boolean propositions.

Definition 4 (Instantiation sequence). An *instantiation sequence* is a quadruple $\langle E, K, S_E, S_K \rangle$, where E is a finite sequence of distinct time points in $\mathcal{T}$, K is a finite sequence of distinct propositions in $\mathcal{P}$, and S_E and S_K are partial schedules whose domains are E and K, respectively. □

Definition 5 (Execution sequence). An *execution sequence* $Z = \langle E, K, S_E, S_K \rangle$ is an instantiation sequence satisfying the following properties:

- S_E **Monotonicity:** For any pair $X_i, X_j \in E$, if $i < j$, then $S_E(X_i) \leq S_E(X_j)$.
- **Time Point Label Honesty:** For each $X \in E$ and each literal $\lambda \in L(X)$ where $\lambda \in \{p, \neg p\}$, it holds that $O(p) \in E$ and $O(p)$ is before X, $p \in K$, $S_K(p) = \top$ (if $\lambda = p$) and $S_K(p) = \bot$ (if $\lambda = \neg p$). Also, $S_E(O(p)) < S_E(X)$ (if $p \in \mathcal{OP}$) and $S_E(O(p)) \leq S_E(X)$ (if $p \in \mathcal{DP}$).

Z^* denotes the set of all execution sequences, $t_{last}(Z) = \max\{S_E(X) \mid X \in E\}$ denotes the last time instant in which a time point was executed in Z, and $last(Z) = \{X \mid X \in E \wedge S_E(X) = t_{last}\}$ denotes the set of the last executed time points. □

Therefore, an execution sequence models the ordered sequence of executed time points and assigned propositions according to the well-definedness of a CSTNUD.

Example 4.1. Consider Figure 4b. Assume that we execute S at 0, `ProcOS` at 1, `ProcOE` at 2, $O?$ at 3 and observe $\neg o$. Assume also that we execute `NormCS` at 4 and observe `NormCE` to occur at 24 (i.e., at its maximal duration). The execution sequence is $Z = \langle \langle S, \mathtt{ProcOS}, \mathtt{ProcOE}, O?, \mathtt{NormCS}, \mathtt{NormCE} \rangle, \langle o \rangle,$

$\{S_E(S) = 0, S_E(\texttt{ProcOS}) = 1, S_E(\texttt{ProcOE}) = 2, S_E(O?) = 3, S_E(\texttt{NormCS}) = 4, S_E(\texttt{NormCE}) = 24\}, \{S_K(o) = \bot\}\rangle$. □

We can now first compute the current partial scenario as the conjunction of all positive and negative literals arising from all propositions in K according to S_K and then define local consistency.

Definition 6 (Current partial scenario). For any $Z = \langle E, K, S_E, S_K\rangle$, the *current partial scenario* is given by $\ell_{cps} = \lambda_1 \wedge \cdots \wedge \lambda_k$, where for each $p_i \in K$, it holds that $\lambda_i = p_i$ if $S_K(p_i) = \top$ and $\lambda_i = \neg p_i$ if $S_K(p_i) = \bot$. □

Example 4.2. For Z in our running example, we have that $\ell_{cps} = \neg o$ since $o \in K$ and $S_K(o) = \bot$. □

Definition 7 (Local consistency). An execution sequence $Z = \langle E, K, S_E, S_K\rangle$, is *locally consistent* if and only if for each $(Y - X \leq k, \ell) \in \mathcal{C}$, where $X, Y \in E$ and $\ell_{cps} \Rightarrow \ell$, it holds that $S_E(Y) - S_E(X) \leq k$. □

Example 4.3. In our running example, Z is locally consistent since the schedule S_E satisfies $(S - \texttt{ProcOS} \leq -1, \boxdot)$, $(\texttt{ProcOE} - \texttt{ProcOS} \leq 2, \boxdot)$, $(\texttt{ProcOS} - \texttt{ProcOE} \leq -1, \boxdot)$, $(O? - \texttt{ProcOE} \leq 1, \boxdot)$, $(\texttt{ProcOE} - O? \leq -1, \boxdot)$, $(\texttt{NormCS} - O? \leq 1, \neg o)$ and $(O? - \texttt{NormCS} \leq -1, \neg o)$. □

An execution sequence evolves over time according to the evolution of the game that `Player1` (the controller) plays against `Player2` (the environment). Each player follows a strategy saying what moves to make and when. Moreover, many moves can be made at the same time instant (provided that they respect an order) and sometimes moves are mandatory.

Definition 8 (Move). A *move* m is either X *meaning "execute time point* X*"* or $(p, \texttt{b})$ *meaning "assign* $\texttt{b} \in \{\top, \bot\}$ *to proposition* p*"*. A move for the controller `Player1` requires that X is a *non-contingent* time point and p is a *decidable* proposition. A move for the environment `Player2` requires that X is a *contingent* time point and p is an *observable* proposition. M_1^* and M_2^* denote the sets of all moves for `Player1` and `Player2`, respectively. □

A *move-based strategy* is a mapping from execution sequences considered at particular time instants to moves augmented with a `wait` condition modeling the absence of move. A strategy tells a player to make a move at a particular time instant only if the move is applicable at that particular time. Therefore, a strategy specifies *applicability conditions* saying when a move *can* be made, *obligations* saying when a move *has to* be made and *postconditions* saying how the execution sequence evolves accordingly.

Definition 9 (Move-based strategy). A *move-based strategy* for the controller `Player1` is a mapping $\sigma_1 \colon Z^* \times \mathbb{R} \to M_1^* \cup \{\texttt{wait}\}$ such that its *applicability conditions are*:

(1) For any execution sequence Z and any time instant t, $\sigma_1(Z,t)$ is *applicable* iff $t \sim t_{last}(Z)$, where $\sim$ is $>$ if $last(Z)$ contains a contingent time point C or an observation time point $P?$ such that K contains its related proposition p (*reaction time enforcement*), and $\sim$ is $\geq$ otherwise.
(2) For any execution sequence Z and any time instant t, $\sigma_1(Z,t) = X$ is applicable if condition (1) holds and X is an unexecuted non-contingent time point such that the current partial scenario entails $L(X)$ (i.e., $X \notin E \wedge X \notin \mathit{Contingent} \wedge \ell_{cps} \Rightarrow L(X)$).
(3) For any execution sequence Z and any time instant t, $\sigma_1(Z,t) = \texttt{wait}$ is applicable if (1) holds and there is no obligation at time t.

The unique *obligation* involves decidable propositions requiring that whenever a decision time point $D!$ has been executed and its related proposition d has not been assigned yet, then the strategy must issue a move to assign d a truth value instantaneously. That is,

$$D! \in E \wedge d \notin K \implies \sigma_1(Z, S_E(D!)) = (d, \texttt{b})$$

for any decision time point $D! \in \mathcal{DT}$.

A *move-based strategy* for the environment `Player2` is a mapping $\sigma_2 \colon Z^* \times \mathbb{R} \to M_2^* \cup \{\texttt{wait}\}$ such that its *applicability conditions* are:

(1) For any execution sequence Z and any time instant t, $\sigma_2(Z,t)$ is *applicable* iff $t \geq t_{last}(Z)$ (no reaction time enforcement needed).
(2) For any execution sequence Z, any time instant t and any contingent link $(A, x, y, C) \in \mathcal{L}$, $\sigma_2(Z,t) = C$ is applicable if condition (1) holds, A has already been executed, C has not, and executing C at this time satisfies $C - A \in [x, y]$ (i.e., $A \in E \wedge C \in \mathit{Contingent} \wedge C \notin E \wedge t - S_E(A) \in [x, y]$).

(3) For any execution sequence Z and any time instant t, $\sigma_2(Z,t) = \texttt{wait}$ is applicable if (1) holds and there is no obligation at time t.

Obligations are of two kinds. The first kind of obligation involves observable propositions requiring that whenever an observation time point $P?$ has been executed and its related proposition p has not been assigned yet, then the strategy must issue a move to assign p a truth value instantaneously. That is,

$$(P? \in E \wedge p \notin K) \implies \sigma_2(Z, S_E(P?)) = (p, \texttt{b})$$

for any observation time point $P? \in \mathcal{OT}$.

The second kind of obligation involves contingent links (A, x, y, C) requiring that if A has already been executed, C has not and the current time t is the last instant at which C can be executed, then the strategy must issue a move to execute C at t. That is,

$$(A \in E \wedge C \notin E \wedge t - S_E(A) = y) \implies \sigma_2(Z,t) = C\,.$$

for any $(A, x, y, C) \in \mathcal{L}$ and any real time instant $t \in \mathbb{R}$.

Postconditions for both σ_1 and σ_2 are the same. If the strategy tells the player to execute a time point X at time t, then Z updates by appending X to E and extending S_E such that $S_E(X) = t$. If the strategy tells the player to assign the truth value $\texttt{b}$ to the proposition p, then Z updates by appending p to K and extending S_K such that $S_K(p) = \texttt{b}$. That is,

- If $\sigma_i(Z,t) = X$, then $Post(Z, \sigma_i, t) = \langle E \cup \langle X\rangle, K, S_E \cup \{S_E(X) = t\}, S_K\rangle$.
- If $\sigma_i(Z,t) = (p, \texttt{b})$, then $Post(Z, \sigma_i, t) = \langle E, K \cup \langle p\rangle, S_E, S_K \cup \{S_K(p) = \texttt{b}\}\rangle$. □

Example 4.4. In our running example, we have that $t_{last}(Z) = 24$ and $last(Z) = \{\texttt{NormCE}\}$. Suppose that the current time is $t = 25$. Then $\sigma_1(Z, 25) = O_E$ is applicable since $t > t_{last}$ and O_E has not been executed yet. Now, we have that $t_{last}(Z) = 25$ and $last(Z) = \{O_E\}$. Suppose that current time is $t = 26$. $\sigma_1(Z, 26) = H!$ is applicable, whereas $\sigma_1(Z, 26) = (h, \top)$ is not since $H! \notin E$. If $\sigma_1(Z, 26) = H!$ is taken into consideration (i.e., $Z' = Post(Z, \sigma_1, t)$), then $\sigma_1(Z', 26) = (h, \top)$ is instantaneously issued after. □

We model `Player2` as the *most powerful player possible*. If `Player1` can beat this worst-case environment, then `Player1` must be able to beat any other less powerful environment playing the same game. To achieve this purpose, we model the game to proceed in *turns*. That is, at any time instant t, there exist two turns: $T_1(t)$ (occurring first) and $T_2(t)$ (occurring last). `Player1` makes his moves in $T_1(t)$, whereas `Player2` makes his in $T_2(t)$. If player i does not make any move in $T_i(t)$, then he loses forever the possibility to play at time t. As a result, `Player2`, making his moves in $T_2(t)$, is guaranteed to always have full information on what `Player1` has done in $T_1(t)$ (which corresponds to the worst-case scenario).

In what remains of this section we define the notions of *snapshot* modeling an execution sequence at a particular time instant t (after the players are done in $T_1(t)$ and $T_2(t)$), *continuous game evolution* modeling how the execution sequence evolves and *winning conditions* for each player.

Definition 10 (Snapshot). Let $Z = \langle E, K, S_E, S_K \rangle$ be an execution sequence. $Z(t) = \langle E', K', S'_E, S'_K \rangle$ models the *snapshot* of Z at time t, where $E' = \langle X \mid X \in E \wedge S_E(X) \leq t \rangle$[3], $K' = \langle p \mid p \in K \wedge O(p) \in E' \rangle$, $S'_E(X) = S_E(X)$ for all $X \in E'$, and $S'_K(p) = S_K(p)$ for all $p \in K'$. □

Example 4.5. Consider again our running example, i.e., the execution sequence we discussed above. At $t = 1$, we have $Z = \langle \langle S, \texttt{ProcOS} \rangle, \emptyset, \{S_E(S) = 0, S_E(\texttt{ProcOS}) = 1\}, \emptyset \rangle$. At $t = 2$, we have $Z = \langle \langle S, \texttt{ProcOS}, \texttt{ProcOE} \rangle, \emptyset, \{S_E(S) = 0, S_E(\texttt{ProcOS}) = 1, S_E(\texttt{ProcOE}) = 2\}, \emptyset \rangle$ and so on. □

Definition 11 (Continuous game evolution). Let $t \in \mathbb{R}^{\geq 0}$ be the global time. The *continuous game evolution* is modeled by an infinite sequence of snapshots $Z(t)$ defined as:

$$Z(t) = \begin{cases} T_2(T_1(\langle \emptyset, \emptyset, \emptyset, \emptyset \rangle, t), t) & \text{if } t = 0 \\ T_2(T_1(Z(t - \epsilon), t), t) & \text{if } t > 0 \end{cases}$$

$$T_i(Z, t) = \begin{cases} Z & \text{if } \sigma_i(Z, t) = \texttt{wait} \\ T_i(Post(Z, \sigma_i, t), t) & \text{otherwise} \end{cases}$$

[3]That is, E' is the subsequence of E up to time point X_i where $S_E(X_i) \leq t'$ (sequence comprehension).

where $T_i(t)$ models the evolution of Z during turn i at time t according to σ_i, whereas an (arbitrary small) $\epsilon > 0$ models the reaction time. □

Definition 12 (Winning conditions). `Player1` wins the game if and only if the game evolution leads to a snapshot $Z(t)$ such that for each unexecuted time point X, ℓ_{cps} falsifies $L(X)$ and for each constraint $(Y - X \leq k, \ell)$, where $X, Y \in E$ and $\ell_{cps} \Rightarrow \ell$, we have that $S_E(Y) - S_E(X) \leq k$. `Player2` wins otherwise. σ_i is a *winning strategy* if player i wins the game by following σ_i. □

Definition 13 (Dynamic controllability). A CSTNUD is *dynamically controllable* if `Player1` has a winning strategy such that for any $t > 0$ and any pair of execution sequences Z_1 and Z_2, if $\sigma_2(Z_1, t') = \sigma_2(Z_2, t')$ for $0 \leq t' < t$, then $\sigma_1(Z_1, t) = \sigma_1(Z_2, t)$. □

In other words, whenever `Player2` has made the same sequence of moves up to time $t - \epsilon$, then `Player1` will make the same move(s) at time t.

5. Dynamic Controllability of CSTNUDs via TGAs

In this section, we extend and optimize the encoding given for CSTNUs in Section 3.

5.1. Extension

Consider, as an example, Figure 5 depicting the encoding of the CSTNUD in Figure 4b. Once again, we have three core locations but this time we borrow a few names from Section 4 and rename them to $\mathtt{T}_1$ (in place of L_1), $\mathtt{T}_2$ (in place of L_0) and `win` (in place of `goal`). $\mathtt{T}_1$ and $\mathtt{T}_2$ model the two turns $T_1(t)$ and $T_2(t)$ when global time is > 0. $\mathtt{T}_2$ is the initial location. The winning path is computed in the same way, only renaming each L_{ℓ_i} to $\mathtt{w}_{\ell_i}$. `gain` and `pass` regulate the turns at any time instant t. We still have a clock `cX` for each $X \in \mathcal{T}$ (considering decision time points too) and a clock `bP` for each $p \in \mathcal{P}$ (considering decidable propositions too).

Let us now discuss how to model the truth value assignment to the decidable propositions. Dually to observable propositions, for each decidable proposition $d \in \mathcal{DP}$ we generate an uncontrollable self-loop transition $\langle \mathtt{T}_1, \mathtt{cD} < \hat{\mathtt{c}} \wedge \mathtt{cD} = 0 \wedge \mathtt{bD} = \hat{\mathtt{c}}, \mathtt{dFalse}, \{\mathtt{bD}\}, \mathtt{T}_1 \rangle$ at $\mathtt{T}_1$ (without resetting $\mathtt{c}_\delta$). If we take this transition, it means that we decide $\neg d$. If we don't, it means

that we decide (actually confirm) d. In the former case, such a transition has to be taken at the same instant at which $D!$ was executed but after $\texttt{exD}$ was taken. In this way, we model "how" to decide the truth values of the propositions in $\mathcal{DP}$. All other transitions remain the same as those given for CSTNUs.

5.2. Optimizations

We have discussed the main extension of the encoding that concerns how we make decisions in our temporal plan. That is, how we assign truth values to the decidable propositions upon the execution of decision time points. Let us now optimize this whole encoding. We refine the guard of each uncontrollable self-loop at $\texttt{T}_1$ by exploiting what we know of the CSTNUD. That is, we extend the guards so that they enforce time point label honesty as well as the partial order among the time points when not ambiguous. This optimization was first discussed in [18] but there it dealt with disjunctive constraints and exploited internal data structures provided by the UPPAAL-TIGA software. Here, we propose a more formal definition avoiding such data structures. Moreover, [18] does not address decisions. As an example of this optimization, consider time points $\texttt{FastCS}$ and $\texttt{NormDS}$ of the CSTNUD in Figure 4b. $L(\texttt{FastCS}) = o$ and $L(\texttt{NormDS}) = \neg h$. Recall that the encoding models o and h as two dedicated clocks $\texttt{bO}$ and $\texttt{bH}$ such that if one of these clocks is equal to (resp., less than) $\hat{\texttt{c}}$, once its related observation or decision time point has been executed, then the related proposition is $\top$ (resp., $\bot$). Moreover, time point label honesty also requires that $O? - \texttt{FastCS} \leq -\epsilon$ (observations) and $H! - \texttt{NormDS} \leq 0$ (decisions).

Therefore, considering the time point label honesty property for CSTNUDs, it is possible to extend the guards of $\texttt{exFCS}$ and $\texttt{exNDS}$ by appending $\texttt{bO} = \hat{\texttt{c}} \wedge \texttt{cO} < \hat{\texttt{c}} \wedge \texttt{cO} > 0$ and $\texttt{bH} < \hat{\texttt{c}} \wedge \texttt{cH} < \hat{\texttt{c}} \wedge \texttt{cH} \geq 0$, respectively. The former models the fact that $\texttt{FastCS}$ must be executed if only if $o = \top$ (i.e., $\texttt{bO} = \hat{\texttt{c}}$), which also implies that $\texttt{FastCS}$ must be executed after $O?$ (i.e., $O?$ have been executed ($\texttt{cO} < \hat{\texttt{c}}$)) and a positive amount of time ϵ has elapsed ($\texttt{cO} > 0$). The latter models the fact that $\texttt{NormDS}$ must be executed if only if $h = \bot$ (i.e., $\texttt{bH} < \hat{\texttt{c}}$), which also implies that $\texttt{NormDS}$ must be executed after $H!$ (i.e., $H!$ have been executed ($\texttt{cH} < \hat{\texttt{c}}$)) either instantaneously or after a positive amount of time has elapsed ($\texttt{cH} \geq 0$).

Definition 14 (Encoding time point label honesty). A *label encoder* is a mapping $L_{enc} \colon \mathcal{T} \to \mathcal{H}(\mathcal{X})$ translating the label of a time point into the

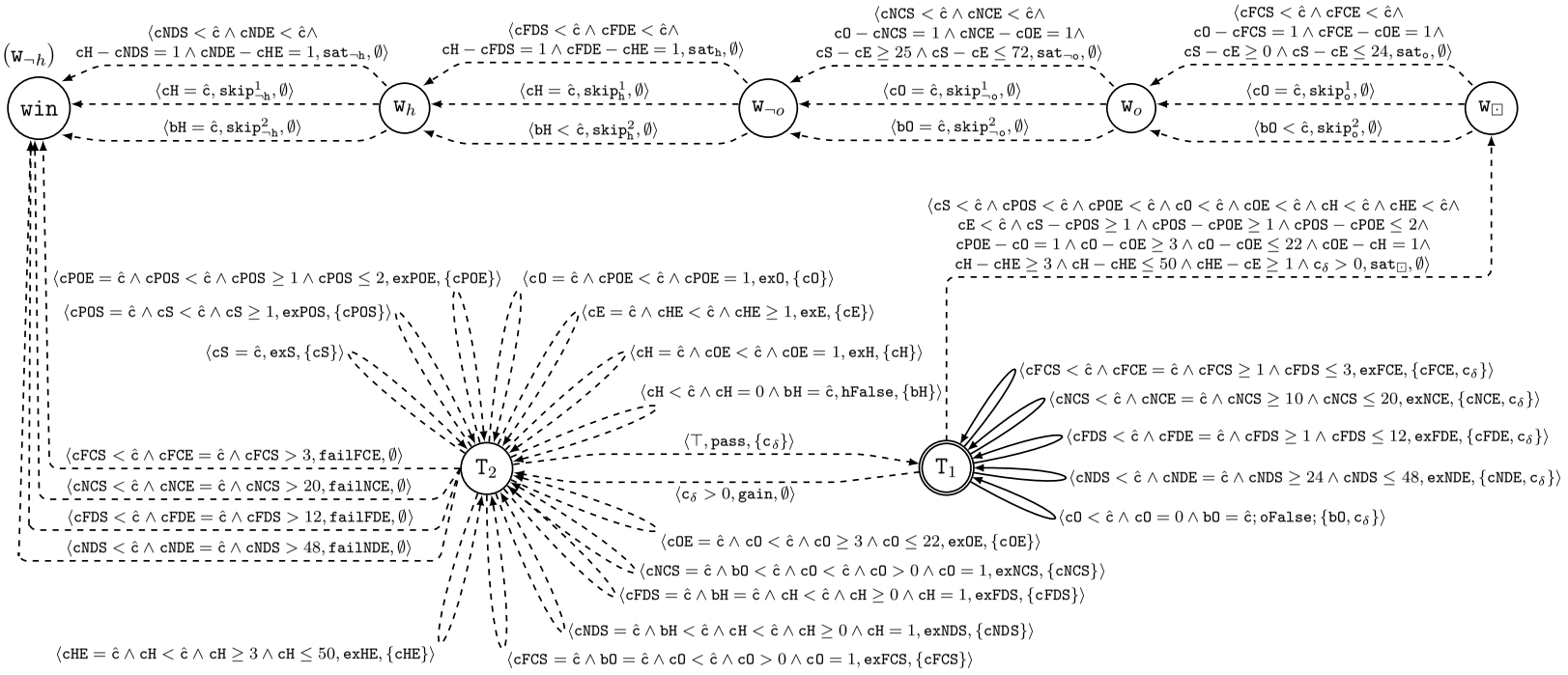


Figure 5: TGA encoding the CSTNUD in Figure 4b. $\mathtt{T}_2$ (ex L_0) is the initial location (modeling $T_2(t)$ for $t > 0$). $\mathtt{T}_1$ (ex L_1) models $T_1(t)$ for $t > 0$). $\mathtt{w}_{\boxdot}$, $\mathtt{w}_o$, $\mathtt{w}_{\neg o}$, $\mathtt{w}_h$, $\mathtt{win}$ (=$\mathtt{w}_{\neg h}$) model the winning path. Some guards have been written in a compact notation (e.g., $\mathtt{c0} \leq 1 \wedge \mathtt{c0} \geq 1$ as $\mathtt{c0} = 1$).

equivalent clock constraint $L_{enc}(X) = L^{\mathcal{OP}}_{enc}(X) \wedge L^{\mathcal{DP}}_{enc}(X)$, where $L^{\mathcal{OP}}_{enc}(X)$ and $L^{\mathcal{DP}}_{enc}(X)$ encode all literals containing observable and decidable propositions, respectively:

- $L^{\mathcal{OP}}_{enc}(X)$: $\bigwedge_{p \in L(X)}(\mathtt{bP} = \hat{\mathtt{c}} \wedge \mathtt{cP} < \hat{\mathtt{c}} \wedge \mathtt{cP} > 0) \bigwedge_{\neg q \in L(X)}(\mathtt{bQ} < \hat{\mathtt{c}} \wedge \mathtt{cQ} < \hat{\mathtt{c}} \wedge \mathtt{cQ} > 0)$.
- $L^{\mathcal{DP}}_{enc}(X)$: $\bigwedge_{d \in L(X)}(\mathtt{bD} = \hat{\mathtt{c}} \wedge \mathtt{cD} < \hat{\mathtt{c}} \wedge \mathtt{cD} \geq 0) \bigwedge_{\neg f \in L(X)}(\mathtt{bF} < \hat{\mathtt{c}} \wedge \mathtt{cF} < \hat{\mathtt{c}} \wedge \mathtt{cF} \geq 0)$. □

We now focus on constraints. Consider the requirement link $O? \rightarrow \mathtt{FastCS}$ labeled by $[1, 1], o$ in the CSTNUD that we are discussing. Such a constraint says that `FastCS` must be executed after 1 and within 1 since $O?$ (thus, exactly after 1 since $O?$). This requirement link has also an important characteristic: $L(\mathtt{FastCS})$ coincides with the label of the link. Therefore, whenever `FastCS` is executed, the constraint must hold. Thus, we extend the original guard of `exFCS` (formerly $\mathtt{cFCS} = \hat{\mathtt{c}}$) not only by adding a conjunction of constraints modeling its label honesty, but also adding $\mathtt{c0} < \hat{\mathtt{c}} \wedge \mathtt{c0} = 1$ stating that $O?$ has already been executed ($\mathtt{c0} < \hat{\mathtt{c}}$) and $\mathtt{FastCS} - O? \in [1, 1]$ ($\mathtt{c0} = 1$). More formally:

Definition 15 (Encoding predecessors). Given a CSTNUD, a *predecessor* of a time point $Y \in \mathcal{T}$ is a time point $X \in \mathcal{T}$ such that there exists a constraint $(X - Y \leq -x, L(Y)) \in \mathcal{C}$, where $x > 0$. $\Pi : \mathcal{T} \rightarrow 2^{\mathcal{T}}$ returns the predecessors of a given time point and it is formalized as $\Pi(Y) = \{X \mid (X - Y \leq -x, \ell) \in \mathcal{C} \wedge x > 0 \wedge \ell = L(Y)\}$. A *predecessor encoder* is a mapping $\Pi_{enc} \colon \mathcal{T} \rightarrow \mathcal{H}(\mathcal{X})$ translating each $X \in \Pi(Y)$ (along with its temporal bounds) into an equivalent clock constraint as follows: $\Pi_{enc}(Y) = \bigwedge_{X \in \Pi(Y)} \mathtt{cX} < \hat{\mathtt{c}} \wedge \mathtt{cX} \geq x \wedge \mathtt{cX} \leq y$, where $\mathtt{cX} \geq x$ models $(X - Y \leq -x, L(Y))$ and $\mathtt{cX} \leq y$ models $(Y - X \leq y, L(Y))$ (if any). □

Therefore, for each non-contingent time point X, the guard of `exX` becomes $\mathtt{cX} = \hat{\mathtt{c}} \wedge L_{enc}(X) \wedge \Pi_{enc}(X)$.

5.3. An optimized encoding for checking DC of CSTNUDs

Let us now join our extension and optimizations to define our full encoding.

Definition 16. Encoding CSTNUDs into TGAs is achieved by extending and optimizing the encoding for CSTNUs discussed in Section 3 to:

(1) model transitions to operate on decidable propositions (Section 5.1),
(2) enforce time point label honesty in the transition guards (Definition 14 in Section 5.2), and
(3) enforce predecessors in the transition guards (Definition 15 in Section 5.2).
□

We point out that (1) serves to adapt the encoding to CSTNUDs, whereas (2) and (3) are optimizations to boost the model checking phase (therefore, they are also applicable to the old encoding for CSTNUs).

6. Correctness and Complexity of the Encoding

In this section, we prove the correctness and discuss the complexity of the encoding provided in Definition 16 in Section 5.

A controllability algorithm for a temporal network is *sound* if whenever the algorithm says "uncontrollable", the temporal network is really uncontrollable and it is *complete* if the algorithm says "uncontrollable" for each uncontrollable temporal network (e.g., see [3]). A controllability algorithm is *correct* if it is sound and complete.

Theorem 1. *The encoding of CSTNUDs into TGAs given in Definition 16 is correct.*

Proof. We begin the proof by showing that the extensions discussed in Section 5 of the encoding for CSTNUs given in Section 3 correctly model the move-based semantics of Section 4. A state of the TGA is given by a pair $(L, \vec{c})$, where L is a location and $\vec{c}$ represents the values of all clocks. The state of a CSTNUD during execution is given by its execution sequence Z. We show that the game interplay correctly models the continuous game evolution given in Definition 11 for all $t > 0$. We exclude the case for $t = 0$, so `Player1` does not play in $T_1(0)$ and `Player2` does not play in $T_2(0)$ either.

(Invariant) At any instant $t > 0$ the snapshot $Z(t) = \langle E, K, S_E, S_K \rangle$ corresponds to a state of the TGA $(L, \vec{c})$ in which $L = \mathtt{T_2}$ and $\vec{c}$ is as follows: $\hat{\mathtt{c}} = t$, $\mathtt{c}_\delta = 0$, for each $X \in \mathcal{T}$, $\mathtt{cX} < \hat{\mathtt{c}}$ and $\hat{\mathtt{c}} - \mathtt{cX} = k$ (if $X \in E \wedge S_E(X) = k$), $\mathtt{cX} = \hat{\mathtt{c}}$ otherwise. For each $p \in \mathcal{P}$, $\mathtt{cP} < \hat{\mathtt{c}} \wedge \mathtt{bP} = \hat{\mathtt{c}}$ (if $p \in K$ and $S_K(p) = \top$) and $\mathtt{cP} < \hat{\mathtt{c}} \wedge \mathtt{bP} < \hat{\mathtt{c}}$ (if $p \in K$ and $S_K(p) = \top$), $\mathtt{cP} = \mathtt{bP} = \hat{\mathtt{c}}$ otherwise. Finally, `Player2` has finished taking controllable transitions at t.

When $t = 0$ (i.e., $\hat{\mathtt{c}} = 0$) `Player2` cannot play in $\mathtt{T}_2$ as no controllable transition is enabled. `Player1` cannot play either because the current location is not $\mathtt{T}_1$ and he can only get there after a positive amount of time has elapsed. Therefore, at $t = 0$, $Z(0) = \langle \emptyset, \emptyset, \emptyset, \emptyset \rangle$.

When $t > 0$ (i.e., $\hat{\mathtt{c}} > 0$) both `Player1` and `Player2` can play in their respective turns $T_1(t)$ and $T_2(t)$. `Player1` can take `gain` to enter $\mathtt{T}_1$ at time t. `Player2` cannot prevent him from doing so because `gain`, being urgent, has priority over any other controllable transition that `Player2` could take at that time. So, `Player1` plays first. Once he has entered $\mathtt{T}_1$, `Player1` can take (in general) a non-empty sequence of transitions to execute a few non-contingent time points and decide the truth values of some decidable propositions (if he has executed some decision time points instantaneously before). Such a sequence is finite since there is a finite number of time points to execute and a finite number of decidable propositions to assign (one for each decision time point). Furthermore, each time point (resp., proposition) can be executed (resp., assigned a value) only once. When this sequence of transitions is over, `Player1` ends his turn by taking `pass` to lead the run back to $\mathtt{T}_2$. Since $\mathtt{T}_1$ is urgent, time has not elapsed. Therefore, the sequence of transitions taken at $\mathtt{T}_1$ corresponds to the sequence of moves made by `Player1` in $T_1(t)$. Instead, if `Player1` wants to wait at time t, he can either take `gain` and `pass` immediately after or just avoid taking `gain`. Now, at $\mathtt{T}_2$, `Player2` does the same for contingent time points and observable propositions if some observation time points have been executed by `Player1` in $T_1(t)$. When `Player2` is done, the sequence of transitions taken models the sequence of moves made in $T_2(t)$. Since `Player2` does not make any other move in $T_2(t)$, $Z(t)$ no longer changes.

`Player1` and `Player2` are driven by their strategies σ_1 and σ_2, which say what moves to make (i.e., transitions to take) in $T_1(t)$ and $T_2(t)$ (i.e., locations $\mathtt{T}_1$ and $\mathtt{T}_2$) at any time t depending on the current Z. The purpose of σ_1 is to keep $Z(t)$ locally consistent, whereas that of σ_2 is the opposite.

The strategies also satisfy their applicability conditions as `Player1` can make his moves in $T_1(t)$ according to σ_1 iff `Player2` has not played yet in $T_2(t)$, whereas `Player2` can make his moves in $T_2(t)$ according to σ_2 iff either `Player1` has not played in $T_1(t)$ or `Player2` is not done in $T_2(t)$. We have already proved that for any $t > 0$, `Player1` plays first.

The strategies satisfy their obligations as each time `Player1` executes a decision time point $D!$, he also assigns the associated decidable proposition d a truth value as well. This occurs at the same time but instantaneously

after the execution of $D!$. `Player1` assigns $\top$ to d by not taking `dFalse` and assigns $\bot$ to d by taking it. If `Player1` takes the transition, then he will never be able to take it again in the same turn or in the following turns (as the guard of `dFalse` invalidates). If he does not, then he will never be able to take `dFalse` in any $T_1(t')$ where $t' > t$. Likewise, σ_2 satisfies its similar obligation for observable propositions. Furthermore, σ_2 also satisfies the obligation regarding contingent time points as the encoding generates a `failC` transition for each contingent time point C (belonging to a $(A, x, y, C) \in \mathcal{L}$) allowing `Player1` to move to `win` if `Player2` does not take `exC` within its maximum upper bound y. Since `Player2` wants to prevent `Player1` from getting to `win`, σ_2 is obliged to schedule C such that $C - A \in [x, y]$.

Both σ_1 and σ_2 satisfy their postconditions: the reset of `cX` clocks says when the time points were executed, whereas the values of `bP` clocks say what truth values the propositions have been assigned. Finally, winning conditions are modeled differently with respect to the player. For `Player1`, they are abstracted as a winning path checking that all time points and constraints whose labels are not falsified by ℓ_{cps} have been executed and satisfied, respectively. For `Player2`, winning conditions correspond to scheduling a contingent time point at a particular time or deciding a truth value for an observable proposition (or any combination of these moves) such that `Player1` is unable to satisfy at least one constraint and ends up blocked somewhere while going through the winning path before entering `win`. □

Theorem 2. *Any CSTNUD can be encoded into a TGA in polynomial time (with respect to the size of the network).*

Proof. Let $\mathcal{S} = \langle \mathcal{T}, \mathcal{OT}, \mathcal{DT}, \mathcal{P}, O, L, \mathcal{L}, \mathcal{C} \rangle$ be any CSTNUD, let $\mathit{Labels} = \{L(X) \mid X \in \mathcal{T}\} \cup \{\ell \mid (Y - X \leq k, \ell) \in \mathcal{C}\}$ be the set of different labels in the CSTNUD and let $\mathit{length}\colon \mathcal{P}^* \to \mathbb{N}$ be the mapping returning the length of a label (i.e., the number of literals), where

$$\mathit{length}(\ell) = \begin{cases} 0 & \text{if } \ell = \boxdot \\ n & \text{if } \ell = \lambda_1 \wedge \cdots \wedge \lambda_n) \end{cases}$$

The encoding generates $3+|\mathit{Labels}|-1$ locations: $\mathtt{T}_1$, $\mathtt{T}_2$, `win` and $(|\mathit{Labels}|-1)$ $\mathtt{w}_{\ell_i}$. Thus, $\mathit{Locations}(\mathcal{S})$ is of order $\mathcal{O}(|\mathit{Labels}|)$.

The encoding also generates a polynomial number of transitions: 2 transitions for the game interplay (`gain` and `pass`), 2 transitions for each observation time point $P?$ (`exP` and `pFalse`), 2 transitions for each decision time

point $D!$ (`exD` and `dFalse`), 2 transitions for each contingent time point C (`exC` and `failC`), 1 transition for each remaining non-contingent time point X (`exX`) and w transitions going from $\mathtt{T}_2$ to `win` (winning path). Since the number of different labels is fixed in the CSTNUD in input, we are left to prove that each set of transitions connecting $\mathtt{w}_{\ell_{i-1}}$ to $\mathtt{w}_{\ell_i}$ is polynomial in the label $\ell_i = \lambda_1 \wedge \ldots \lambda_n$. The encoding generates a set of $1 + 2 \times \mathit{length}(\ell_i)$ transitions: 1 `sat` verifying that all time points labeled by ℓ_i have been executed and all constraints labeled by ℓ_i are satisfied, and $2 \times \mathit{length}(\ell_i)$ `skip` transitions (for each $\lambda \in \ell_i$ we have a `skip` transition testing that the related observation or decision time point has not been executed and another one testing if the literal does not hold (i.e., if $\neg\lambda_i$ holds). Therefore, $w = 1 + \sum_{\ell \in \mathit{Labels} \setminus \{\boxdot\}}(1 + 2 \times \mathit{length}(\ell))$, and consequently the total number of transitions is $\mathit{Transitions}(\mathcal{S}) = 2 + |\mathcal{P}| + |\mathcal{T}| + |\mathit{Contingent}| + w$ which is of order $\mathcal{O}(|\mathcal{P}| + |\mathcal{T}| + |\mathit{Labels}|)$.

Furthermore, the encoders to enforce time point label honesty and the partial order of time points run in polynomial time too. Indeed, for any non-contingent time point X, $L_{enc}(X)$ scans all literals in $L(X)$ once, and $\Pi_{enc}(X)$ scans all constraints in $\mathcal{C}$ once. Therefore, even though a CSTNUD can express an exponential number of constraints (worst case $\mathcal{O}(2^n)$ labeled constraints where $n = |\mathcal{P}|$), the function $\mathit{Locations}(\mathcal{S}) + \mathit{Transitions}(\mathcal{S})$ does not employ any component of the CSTNUD as an exponent. □

7. Expressiveness of CSTNUDs

In this section, we discuss the expressiveness of CSTNUDs by providing a hierarchy of simple temporal networks (three of which are new) and showing that all other subformalisms can be embedded into CSTNUDs. Figure 6 shows the hierarchy, where an edge $A \to B$ means that formalism B embeds formalism A (e.g., STN $\to$ CSTN means that CSTNs can embed STNs).

Defining CSTNUDs as an extension of CSTNUs implicitly results in also defining three new kinds of networks:

(1) *Simple temporal networks with decisions (STNDs)*, where the bottom formalism STN is augmented with decisions.
(2) *Simple temporal networks with uncertainty and decisions (STNUDs)*, where STNUs are augmented with decisions.
(3) *Conditional simple temporal networks with decisions (CSTNDs)*, where CSTNs are augmented with decisions.

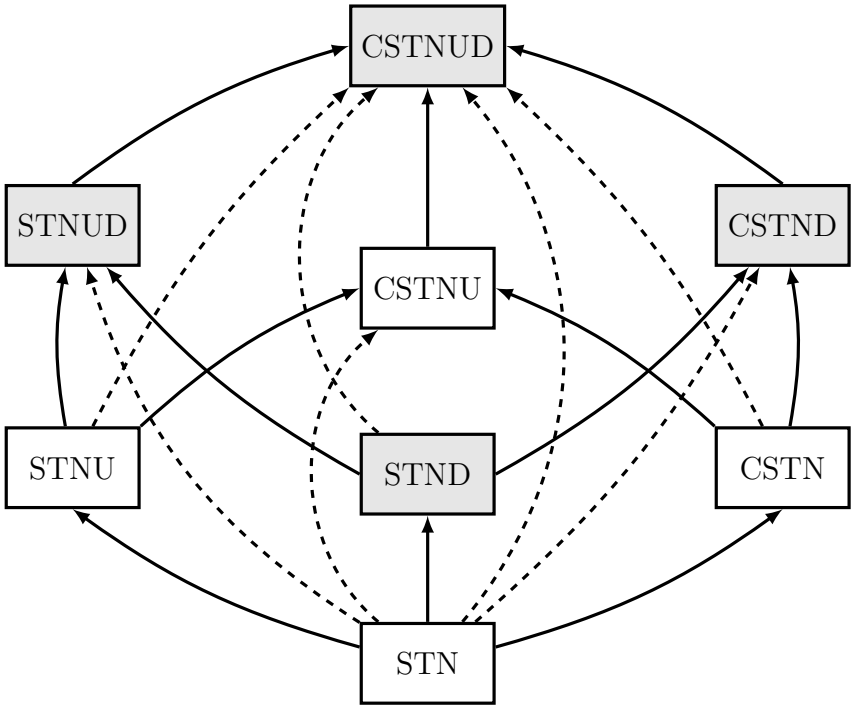


Figure 6: A hierarchy of simple temporal networks. An arrow $A \to B$ means that formalism B embeds formalism A. Solid edges highlight formalisms differing for one extension only (e.g., STNs and STNUs), whereas dashed ones highlight formalisms differing for at least two (e.g., STNs and STNUDs). We highlight new (sub)formalisms as grayed boxes.

We now discuss the encodings from STNs, STNUs, CSTNs, STNDs, STNUDs, CSTNDs and CSTNUs into CSTNUDs.

7.1. Encoding STNs into CSTNUDs

A *Simple Temporal Network* (*STN*, [1]) is a pair $\langle \mathcal{T}, \mathcal{C} \rangle$, where $\mathcal{T}$ is a set of time points and $\mathcal{C}$ is a set of constraints. We encode an STN $\langle \mathcal{T}, \mathcal{C} \rangle$ into a CSTNUD $\langle \mathcal{T}', \mathcal{OT}', \mathcal{DT}', \mathcal{P}', O', L', \mathcal{L}', \mathcal{C}' \rangle$ as follows:

(1) $\mathcal{T}' = \mathcal{T}$.
(2) $\mathcal{OT}' = \mathcal{DT}' = \mathcal{P}' = \mathcal{L}' = \emptyset$.
(3) O is undefined.
(4) $L'(X) = \boxdot$ for each $X \in \mathcal{T}$.
(5) $\mathcal{C}' = \{(Y - X \leq k, \boxdot) \mid (Y - X \leq k) \in \mathcal{C}\}$.

7.2. Encoding STNUs into CSTNUDs

A *Simple Temporal Network with Uncertainty* (*STNU*, [6]) is a triple $\langle \mathcal{T}, \mathcal{L}, \mathcal{C} \rangle$, where $\mathcal{T}$ is a set of time points, $\mathcal{L}$ is a set of contingent links and $\mathcal{C}$ is a set of constraints. We encode an STNU $\langle \mathcal{T}, \mathcal{L}, \mathcal{C} \rangle$ into a CSTNUD $\langle \mathcal{T}', \mathcal{OT}', \mathcal{DT}', \mathcal{P}', O', L', \mathcal{L}', \mathcal{C}' \rangle$ as follows:

(1) $\mathcal{T}' = \mathcal{T}$.
(2) $\mathcal{OT}' = \mathcal{DT}' = \mathcal{P}' = \emptyset$.
(3) $\mathcal{L}' = \mathcal{L}$.
(4) O is undefined.
(5) $L'(X) = \boxdot$ for each $X \in \mathcal{T}$.
(6) $\mathcal{C}' = \{(Y - X \leq k, \boxdot) \mid (Y - X \leq k) \in \mathcal{C}\}$.

7.3. Encoding CSTNs into CSTNUDs

A *Conditional Simple Temporal Network* (*CSTN*, [3]) is a tuple $\langle \mathcal{T}, \mathcal{OT}, \mathcal{P}, O, L, \mathcal{C} \rangle$, where $\mathcal{T}$ is a set of time points, $\mathcal{OT} \subseteq \mathcal{T}$ is a set of observation time points, $\mathcal{P}$ is a set of observable propositions, $O \colon \mathcal{P} \to \mathcal{OT}$ is a bijection assigning a unique decision or observation time point to each proposition and $\mathcal{C}$ is a set of *labeled* constraints. We encode a CSTN $\langle \mathcal{T}, \mathcal{OT}, \mathcal{P}, O, L, \mathcal{C} \rangle$ into a CSTNUD $\langle \mathcal{T}', \mathcal{OT}', \mathcal{DT}', \mathcal{P}', O', L', \mathcal{L}', \mathcal{C}' \rangle$ as follows:

(1) $\mathcal{T}' = \mathcal{T}$.
(2) $\mathcal{OT}' = \mathcal{OT}$.
(3) $\mathcal{P}' = \mathcal{P}$.
(4) $\mathcal{DT}' = \mathcal{L}' = \emptyset$.
(5) $O' = O$.
(6) $L'(X) = L(X)$ for each $X \in \mathcal{T}$.
(7) $\mathcal{C}' = \mathcal{C}$.

7.4. Encoding STNDs into CSTNUDs

A *Simple Temporal Network with Decisions* (*STND*, [9, 10]) is a tuple $\langle \mathcal{T}, \mathcal{DT}, \mathcal{P}, O, L, \mathcal{C} \rangle$, where $\mathcal{T}$ is a set of time points, $\mathcal{DT} \subseteq \mathcal{T}$ is a set of decision time points, $\mathcal{P}$ is a set of decidable propositions, $O \colon \mathcal{P} \to \mathcal{DT}$ is a bijection assigning a unique decision time point to each proposition and $\mathcal{C}$ is a set of *labeled* constraints. We encode an STND $\langle \mathcal{T}, \mathcal{DT}, \mathcal{P}, O, L, \mathcal{C} \rangle$ into a CSTNUD $\langle \mathcal{T}', \mathcal{OT}', \mathcal{DT}', \mathcal{P}', O', L', \mathcal{L}', \mathcal{C}' \rangle$ as follows:

(1) $\mathcal{T}' = \mathcal{T}$.

(2) $\mathcal{DT}' = \mathcal{DT}$.
(3) $\mathcal{P}' = \mathcal{P}$.
(4) $\mathcal{OT}' = \mathcal{L}' = \emptyset$.
(5) $O' = O$.
(6) $L'(X) = L(X)$ for each $X \in \mathcal{T}$.
(7) $\mathcal{C}' = \mathcal{C}$.

7.5. Encoding STNUDs into CSTNUDs

A *Simple Temporal Network with Uncertainty and Decisions* (*STNUD*, [10]) is a tuple $\langle \mathcal{T}, \mathcal{DT}, \mathcal{P}, O, L, \mathcal{L}, \mathcal{C} \rangle$, where $\mathcal{T}$ is a set of time points, $\mathcal{DT} \subseteq \mathcal{T}$ is a set of decision time points, $\mathcal{P}$ is a set of decidable propositions, $O \colon \mathcal{P} \to \mathcal{DT}$ is a bijection assigning a unique decision time point to each proposition, $\mathcal{L}$ is a set of contingent links and $\mathcal{C}$ is a set of *labeled* constraints. We encode an STNUD $\langle \mathcal{T}, \mathcal{DT}, \mathcal{P}, O, L, \mathcal{L}, \mathcal{C} \rangle$ into a CSTNUD $\langle \mathcal{T}', \mathcal{OT}', \mathcal{DT}', \mathcal{P}', O', L', \mathcal{L}', \mathcal{C}' \rangle$ as follows:

(1) $\mathcal{T}' = \mathcal{T}$.
(2) $\mathcal{DT}' = \mathcal{DT}$.
(3) $\mathcal{P}' = \mathcal{P}$.
(4) $\mathcal{OT}' = \emptyset$.
(5) $\mathcal{L}' = \mathcal{L}$.
(6) $O' = O$.
(7) $L'(X) = L(X)$ for each $X \in \mathcal{T}$.
(8) $\mathcal{C}' = \mathcal{C}$.

7.6. Encoding CSTNDs into CSTNUDs

A *Conditional Simple Temporal Network with Uncertainty and Decisions* (*CSTND*, [10]) is a tuple $\langle \mathcal{T}, \mathcal{OT}, \mathcal{DT}, \mathcal{P}, O, L, \mathcal{C} \rangle$, where $\mathcal{T}$ is a set of time points, $\mathcal{OT} \subseteq \mathcal{T}$ is a set of observation time points, $\mathcal{DT} \subseteq \mathcal{T}$ is a set of decision time points, $\mathcal{P}$ is a set of decidable and observable propositions, $O \colon \mathcal{P} \to \mathcal{DT} \cup \mathcal{OT}$ is a bijection assigning a unique decision or observation time point to each proposition and $\mathcal{C}$ is a set of *labeled* constraints. We encode a CSTND $\langle \mathcal{T}, \mathcal{OT}, \mathcal{DT}, \mathcal{P}, O, L, \mathcal{C} \rangle$ into a CSTNUD $\langle \mathcal{T}', \mathcal{OT}', \mathcal{DT}', \mathcal{P}', O', L', \mathcal{L}', \mathcal{C}' \rangle$ as follows:

(1) $\mathcal{T}' = \mathcal{T}$.
(2) $\mathcal{OT}' = \mathcal{OT}$.
(3) $\mathcal{DT}' = \mathcal{DT}$.

(4) $\mathcal{P}' = \mathcal{P}$.
(5) $O' = O$.
(6) $L'(X) = L(X)$ for each $X \in \mathcal{T}$.
(7) $\mathcal{L}' = \emptyset$.
(8) $\mathcal{C}' = \mathcal{C}$.

7.7. Encoding CSTNUs into CSTNUDs

A *Conditional Simple Temporal Network with Uncertainty* (*CSTNU*, [8, 7]) is a tuple $\langle \mathcal{T}, \mathcal{OT}, \mathcal{P}, O, L, \mathcal{L}, \mathcal{C} \rangle$, where $\mathcal{T}$ is a set of time points, $\mathcal{OT} \subseteq \mathcal{T}$ is a set of observation time points, $\mathcal{P}$ is a set of observable propositions, $O \colon \mathcal{P} \to \mathcal{OT}$ is a bijection assigning a unique observation time point to each proposition and $\mathcal{C}$ is a set of *labeled* constraints. We encode a CSTNU $\langle \mathcal{T}, \mathcal{OT}, \mathcal{P}, O, L, \mathcal{L}, \mathcal{C} \rangle$ into a CSTNUD $\langle \mathcal{T}', \mathcal{OT}', \mathcal{DT}', \mathcal{P}', O', L', \mathcal{L}', \mathcal{C}' \rangle$ as follows:

(1) $\mathcal{T}' = \mathcal{T}$.
(2) $\mathcal{OT}' = \mathcal{OT}$.
(3) $\mathcal{DT}' = \emptyset$.
(4) $\mathcal{P}' = \mathcal{P}$.
(5) $O' = O$.
(6) $L'(X) = L(X)$ for each $X \in \mathcal{T}$.
(7) $\mathcal{L}' = \mathcal{L}$.
(8) $\mathcal{C}' = \mathcal{C}$.

Mutatis mutandis, similar encodings apply between all other formalisms $A \to B$ in Figure 6 connected by both dashed and solid arrows.

8. Esse: A tool for CSTNUDs

We developed Esse, a tool for CSTNUDs that takes in input the specification of a CSTNUD and acts both as a solver for dynamic controllability and as an executor simulator. Esse is available at `http://regis.di.univr.it/Esse_TCS.tar.bz2` along with the case studies discussed in this paper and another 1000 randomly generated CSTNUDs as an initial set of benchmarks. Esse is a software that completely supersedes the first prototype provided in

[10], although, for comparison purposes, we also compiled a different version of Esse implementing the old UPPAAL-TIGA encoding.[4]

Esse relies on UPPAAL-TIGA for the model checking phase, but it differs from the previous prototype for two main reasons. First, it makes the interaction with UPPAAL-TIGA transparent. We no longer need to run UPPAAL-TIGA manually as Esse handles it internally. Second, the new UPPAAL-TIGA encoding is optimized by exploiting Boolean variables (instead of clocks) to model propositions and organizing all clocks in array data structures. These two modifications result in the model checking phase performing better and Esse becoming usable also for designers with little or no knowledge of TGAs. Listing 1 shows Esse's help screen.

Listing 1: Esse's help screen.

```
Usage: java -jar esse.jar <Network.cstnud> <Action> dynamic <Network.s> [N] [--silent]

<Action>:
  --check    internally encodes the CTSNUD in input into an UPPAAL-TIGA specification ready
       to
              check dynamic controllability (saves the strategy to Network.s)

  --execute   performs [N] (default 1) executions of the CSTNUD in input (if controllable)
              according to the strategy (.s) synthesized by UPPAAL-TIGA.

--silent    suppresses output (optional)

Examples:
  java -jar esse.jar Network.cstnud --check dynamic Network.s
  java -jar esse.jar Network.cstnud --execute dynamic Network.s 10
```

Given a specification of a CSTNUD, we can check the CSTNUD's dynamic controllability and then carry out `N` execution simulations (if DC) by running

```
$ java -jar esse.jar Network.cstnud --check dynamic Network.s
$ java -jar esse.jar Network.cstnud --execute dynamic Network.s N
```

Listing 2 shows the specification of Figure 4b into Esse's input language. We omit the specifications of Figure 4a and Figure 4c as they are very similar.

Listing 2: Specification of Figure 4b.

[4]Due to a few coding issues, it was not possible to use directly the old prototype in the environment that we set up for the experimental evaluation. In order to make the comparison we thus compiled a different version of Esse supporting the old approach.

```
Propositions {
       o h
}

TimePoints {
       (S : )
       (ProcOS : )
       (ProcOE : )
       (O? : o : )
       (FastCS : o)
       (FastCE : o)
       (NormCS : !o)
       (NormCE : !o)
       (OE : )
       (H! : h : )
       (FastDS : h)
       (FastDE : h)
       (NormDS : !h)
       (NormDE : !h)
       (HE : )
       (E : )
}

ContingentLinks {
       (FastCS,1,3,FastCE)
       (NormCS,10,20,NormCE)
       (FastDS,1,12,FastDE)
       (NormDS,24,48,NormDE)
}

Constraints {
       (S - ProcOS <= -1 : )
       (ProcOE - ProcOS <= 2 : )
       (ProcOS - ProcOE <= -1 : )
       (O - ProcOE <= 1 : )
       (ProcOE - O <= -1 : )
       (FastCS - O <= 1 : o)
       (O - FastCS <= -1 : o)
       (NormCS - O <= 1 : !o)
       (O - NormCS <= -1 : !o)
       (OE - FastCE <= 1 : o)
       (FastCE - OE <= -1 : o)
       (OE - NormCE <= 1 : !o)
       (NormCE - OE <= -1 : !o)
       (H - OE <= 1 : )
       (OE - H <= -1 : )
       (FastDS - H <= 1 : h)
       (H - FastDS <= -1 : h)
       (NormDS - H <= 1 : !h)
       (H - NormDS <= -1 : !h)
       (HE - FastDE <= 1 : h)
       (FastDE - HE <= -1 : h)
       (HE - NormDE <= 1 : !h)
       (NormDE - HE <= -1 : !h)
       (E - S <= 24 : o)
       (S - E <= 0 : o)
       (E - S <= 72 : !o)
```

```
58        (S - E <= -25 : !o)
59        (OE - O <= 22 : )
60        (O - OE <= -3 : )
61        (HE - H <= 50 : )
62        (H - HE <= -3 : )
63        (HE - E <= -1 : )
64  }
```

A specification comprises the following four main sections.

The section `Propositions`

```
Propositions {
  p ... r
}
```

specifies the set $\mathcal{P}$, here comprising of `p, ..., r` as an example.

The section `TimePoints`

```
TimePoints {
  ...
  (D! : d : p !q ...)
  (P? : p : !q ...)
  (X :  : p !q ...)
  ...
}
```

specifies the sets $\mathcal{T}$, $\mathcal{OT}$ and $\mathcal{DT}$ as well as the mappings O and L, and provides here examples of the specification of: a decision time point $D!$ with $d = O(D!)$ and $L(D!) = p\neg q \ldots$, an observation time point $P?$ with $p = O(P?)$ and $L(D!) = \neg q \ldots$ (note the substitution of `?` for `!`), and a general time point $X \notin \mathcal{DT} \cup \mathcal{OT}$ with $L(X) = p\neg q$.

The section `ContingentLinks`

```
ContingentLinks {
  ...
  (A1,10,20,C1)
  ...
}
```

specifies the set $\mathcal{L}$ and provides here an example of the specification of a contingent link $(A_1, 10, 20, C_1) \in \mathcal{L}$.

Finally, the section `Constraints`

```
Constraints {
  ...
  (X - Y <= 10 : !d)
  ...
}
```

specifies the set $\mathcal{C}$ and provides here an example of the specification of a constraint $(X - Y \leq 10, !d) \in \mathcal{C}$.

We ran Esse on the specifications of Figure 2, Figure 4a, Figure 4b and Figure 4c to check whether or not they were dynamically controllable. We used a Linux virtual machine run on top of a VMWare ESXi hypervisor using a physical machine equipped with an Intel i7 2.80GHz and 20GB of RAM for the experimental evaluation. The VM was assigned full CPU power and 16GB of RAM (which is plenty, since UPPAAL-TIGA is provided for 32bit architectures). For the CSTNU in Figure 2 the analysis took 14 minutes and 51 seconds, whereas for each network in Figure 4 the analysis took about 15 minutes and 2 seconds . For Figure 2 and Figure 4b the analysis answered `Uncontrollable`, for Figure 4b the analysis answered `Controllable` saving a 167-action strategy for `Player1` of 512KB, and for Figure 4c the analysis answered `Controllable` too saving a 128-action strategy for `Player1` of 448KB, as shown in Listing 3 (validate once, execute anytime).

Listing 3: Example of DC-checking for Figure 4a, Figure 4b and Figure 4c.

```
$ java -jar esse.jar Figure2.cstnu --check dynamic Figure2.s
Checking Figure2.cstnu with UPPAAL-TIGA
Running UPPAAL-TIGA ...
Uncontrollable (strategy for Player2 exists)

$ java -jar esse.jar Figure4a.cstnud --check dynamic Figure4a.s
Checking Figure4a.cstnud with UPPAAL-TIGA
Running UPPAAL-TIGA ...
Uncontrollable (strategy for Player2 exists)

$ java -jar esse.jar Figure4b.cstnud --check dynamic Figure4b.s
Checking Figure4b.cstnud with UPPAAL-TIGA
Running UPPAAL-TIGA ...
Saving a 167-action strategy to Figure4b.s

$ java -jar esse.jar Figure4c.stnud --check dynamic Figure4c.s
Checking Figure4c.stnud with UPPAAL-TIGA
Running UPPAAL-TIGA ...
Saving a 128-action strategy to Figure4c.s
```

The same analyses would have gone out of memory using the old UPPAAL-TIGA encoding.[5] Therefore, Esse proved to be more scalable than the previous prototype.

We carried out 10,000 dynamic execution simulations of the CSTNUD in Figure 4b using the synthesized strategy `Figure4b.s`. In each execution, Esse randomly assigned a truth value to o upon the execution of O? and real

[5]Moreover, we noted that the performance of the model checking phase also depends on the order of the statements (clocks in particular) in the UPPAAL-TIGA specification. Two specifications defining the same clocks in different orders might perform differently.

values to the contingent links that turned relevant for o and h. No execution crashed. Listing 4 shows a few execution simulations in which we isolated the scenarios of interest to prove that `hurry!` is made dynamically.

Listing 4: Three random executions for Figure 4b

```
1  $ java -jar esse.jar DP.cstnud --execute dynamic DP.s 10000
2  Execution 1         |     Execution 2          |     Execution 3
3  ------------------------------------------------------------------------
4  S = 0.1             |     S = 0.1              |     S = 0.1
5  ProcOS = 1.1        |     ProcOS = 1.1         |     ProcOS = 1.1
6  ProcOE = 2.1        |     ProcOE = 2.1         |     ProcOE = 2.1
7  O = 3.1, o = true   |     O = 3.1, o = false   |     O = 3.1, o = false
8  FastCS = 4.1        |     NormCS = 4.1         |     NormCS = 4.1
9  FastCE = 6.1        |     NormCE = 23.5        |     NormCE = 18.6
10 OE = 7.1            |     OE = 24.5            |     OE = 19.6
11 H = 8.1, h = true   |     H = 25.5, h = true   |     H = 20.6, h = false
12 FastDS = 9.1        |     FastDS = 26.5        |     NormDS = 21.6
13 FastDE = 21.1       |     FastDE = 33.1        |     NormDE = 47.0
14 HE = 22.1           |     HE = 34.1            |     HE = 48.0
15 E = 23.1            |     E = 35.1             |     E = 49.0
16 Verifying ... SAT!  |     Verifying ... SAT!   |     Verifying ... SAT!
```

In all executions `ProcO` lasts exactly 1, starts at 1.1 and ends at 2.1 (lines 5,6). In the first execution, the handled order requires a one-day delivery (line 7) so `FastC` starts at 4.1 and is observed to end at 6.1 (lines 8,9). `FastD` is chosen, starts at 9.1 and is observed to end at 21.1 (lines 12,13). The process completes in time at 23.1 (line 15). In the second execution, the handled order does not require a one-day delivery so `NormC` starts at 4.1 and is observed to end at 23.5. Then, `FastD` is chosen (and not `NormD` as this type of delivery could lasts 48 hours making the process terminate at 76.5), starts at 26.5 and is observed to end at 33.1. The process completes in time at 35.1. In the third execution, the handled order does not require a one-day delivery either so `NormC` starts at 4.1 and is observed to end at 18.6. This time `NormD` is chosen (and not `FastD` as this type of delivery could last 1 hour making the process terminate at 24.6), starts at 21.6 and is observed to end at 47.0. The process completes in time at 49.

The development of Esse allowed us to carry out again the experimental evaluation discussed in [10] and verify performance improvements. Each CSTNUD in that set has 5 to 20 time points, max $\lfloor \frac{|\mathcal{T}|}{5} \rfloor$ contingent links, max $\lfloor \frac{|\mathcal{T}|}{5} \rfloor$ observation time points, max $\lfloor \frac{|\mathcal{T}|}{5} \rfloor$ decision time points, a number of constraints obtained as the floor of 5 to 10% of $|\mathcal{T}|^2 - 2 \times |\mathcal{L}|$. The CSTNUDs were generated by trying to maximize these numbers which looked like a good setting to avoid both under-constrained and over-constrained networks (resulting in a concrete chance of getting both controllable and uncontrollable

CSTNUDs). Algorithm 1 shows the pseudo code of the generator, which was embedded in the first prototype in [10] and used to generate the set of benchmarks (Esse does not contain any generator at the moment).

We ran the analysis on this set by imposing a time out of 900 seconds on each instance. The analysis proved that 326 networks were DC and 27 non-DC. The remaining networks reached the timeout limit. We executed each CSTNUD proved dynamically controllable 1000 times by randomly assigning truth values to observable propositions and durations to contingent links. No execution crashed.

Figure 7a shows the DC-checking on the whole set of benchmarks comparing the performance of Esse with the old experimental evaluation in [10]. Figure 7b shows the space consumed by the strategies related to those CSTNUDs proved dynamically controllable. We can see that strategies synthesized by Esse take more disk space than those synthesized by UPPAAL-TIGA (average space of Esse's is 912.63KB, average space of UPPAAL-TIGA's is 351.83KB). This is because UPPAAL-TIGA returns a strategy in text format that is not directly usable, whereas Esse translates it into a dedicated data structure of objects ready for the execution. Figure 7c shows how long it takes to load the strategies in memory and then carry out 1000 execution simulations.

9. Related Work

In the introduction, we discussed the main extensions of simple temporal networks. In this section, we focus on the main differences between those (and other) formalisms and the one proposed in this paper.

An *STN* [1] is able to model a temporal plan in which it is possible to constrain the distance between pairs of events and the occurrence of all events is under the control of the executing agent (i.e., a real-time planner). For each pair of events, the temporal distance can be limited to stay in a range of real values. Consistency analysis is able to determine whether it is possible to schedule events such that all the given temporal constraints are not violated. The decision problem of consistency for STNs has polynomial-time complexity [1].

Drake [2] is an executive for temporal plans with choices modeled as *Labeled STNs*. Consistency analysis extends to find also suitable values for these choices such that the resulting projected temporal plan is consistent.

Algorithm 1: `CSTNUD-Gen`$(nTPs, maxObs, maxDec, maxCL, maxConstr)$

Input: The number of time points ($nTPs$), the maximum number of observation time points ($maxObs$), the maximum number of decision time points ($maxDec$), the maximum number of contingent links ($maxCL$), and the maximum number of constraints ($maxConstr$).

Output: A well-defined CSTNUD with at least 1 observation, 1 decision, 1 contingent link and such that each proposition labels some component. That is, a *real* CSTNUD.

Check that $(maxObs + maxDec + (2 * maxCL)) \leq nTPs$ ▷ it must be possible

$\mathcal{Z} \leftarrow \langle \mathcal{T}, \mathcal{OT}, \mathcal{DT}, \mathcal{P}, O, L, \mathcal{L}, \mathcal{C} \rangle$ ▷ Empty CSTNUD

▷ Final cardinality of sets $\mathcal{OT}$, $\mathcal{DT}$ and $\mathcal{L}$

$nObs \leftarrow Random(1, maxObs)$, $nDec \leftarrow Random(1, maxDec)$, $nCL \leftarrow Random(1, maxCL)$

▷ Generate observation time points and related propositions

$\mathcal{OT} \leftarrow \{P_i? \mid 1 \leq i \leq nObs\}$, $\mathcal{P} \leftarrow \{p_i? \mid 1 \leq i \leq nObs\}$, $O(p_i) = P_i?$ for $1 \leq i \leq nObs$, $\mathcal{T} \leftarrow \mathcal{OT}$

▷ Generate decision time points and related propositions

$\mathcal{DT} \leftarrow \{D_i! \mid 1 \leq i \leq nDec\}$, $\mathcal{P} \leftarrow \mathcal{P} \cup \{d_i? \mid 1 \leq i \leq nDec\}$, $O(d_i) = D_i!$ for $1 \leq i \leq nDec$, $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{DT}$

▷ Generate contingent links where each $x_i = Random(1, 10)$ and $y_i = Random(11, 20)$

$\mathcal{L} \leftarrow \{(A_i, x_i, y_i, C_i) \mid 1 \leq i \leq nCL\}$, $\mathcal{T} \leftarrow \mathcal{T} \cup \{A, C \mid (A, x, y, C) \in \mathcal{L}\}$

▷ Generate the rest of time points

$\mathcal{T} \leftarrow \mathcal{T} \cup \{X_i \mid 1 \leq i \leq (nTPs - nObs - nDec - (2 * nCL))\}$

$p \leftarrow$ *Random* element in $\mathcal{P}$

$P = O(p)$, $L(P) = \boxdot$ ▷ One decision/observation must be unlabeled

▷ Label time points as follows

$Unlabeled \leftarrow [\mathcal{OT}, \mathcal{DT}, \{\mathcal{T} \setminus \mathcal{OT} \setminus \mathcal{DT} \setminus \{C \mid (A, x, y, C) \in \mathcal{L}\}]$ ▷ List

Remove P from $Unlabeled$ ▷ already assigned $\boxdot$

$maxLength \leftarrow Random(0, |\mathcal{P}|)$ ▷ Max length of a label

while $Unlabeled \neq \emptyset$ **do**

  $X \leftarrow$ pop first element from $Unlabeled$

  $tmp \leftarrow$ Random sample of $\mathcal{P}$ of size $maxLength$

  $L(X) = \boxdot$

  while $tmp \neq \emptyset$ **do**

    $p \leftarrow$ pop a proposition from tmp and decide a sign for it

    $P \leftarrow O(p)$

    if *L(P) has already been specified and* $P \neq X$ *and* $L(X) \wedge L(P) \wedge (\neg)p$ *is consistent* **then**

      $L(X) \leftarrow L(X) \wedge L(P) \wedge (\neg)p$ **if** X *is an activation time point* **then**

        $L(C) = L(X)$ where C is the associated contingent

▷ Generate constraints

for $i \leftarrow 1$ *to* $maxConstr$ **do**

  $X, Y \leftarrow$ two Random time points in $\mathcal{T}$

  Check that $X \neq Y$ and X, Y do not specify a contingent link

  $\ell \leftarrow L(X) \wedge L(Y)$

  Randomly decide to extend ℓ ▷ Probability of this choice is $\frac{1}{2}$

  if *so* **then**

    Get a sample of propositions from $\mathcal{P}$ not appearing in ℓ and try to extend ℓ the same way as we did with time points

  $k \leftarrow Random(-100, 100)$

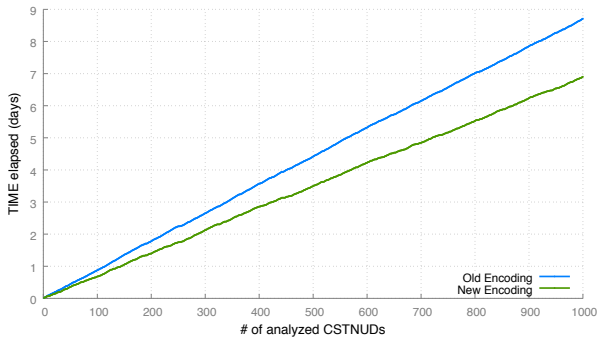
  Add $(Y - X \leq k, \ell)$ to $\mathcal{C}$

▷ Final check

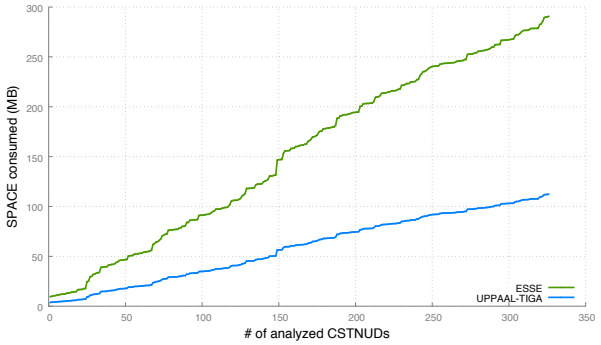
if *Some proposition never appears in any label* **then**

  Throw away the network

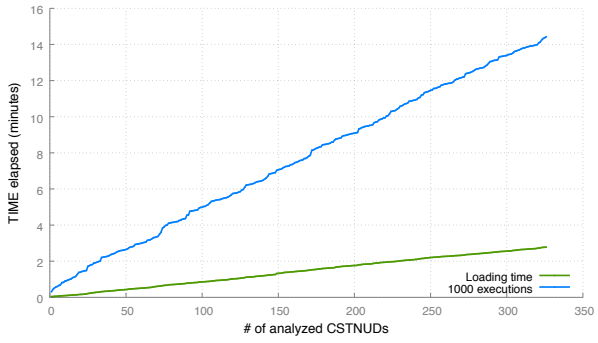
return $\mathcal{Z}$



(a) Dynamic controllability checking on all CSTNUDs with the old encoding (blue, above) and the new one (green, below).



(b) Strategy space consumed by UPPAAL-TIGA (blue, below) and Esse (green, above) for dynamically controllable CSTNUDs.



(c) Strategy loading time (green, below), then 1000 executions (blue, above).

Figure 7: Time and space analysis with Esse on 1000 CSTNUDs.

Consistency is checked by providing an extension of the Bellman-Ford algorithm.

Temporal Plan Networks (*TPNs*, also known as *temporal planning networks*, [19]) rely on a structured approach and can model temporal plans with controllable choices. TPNs handle conditional paths depending on which outgoing edge of a decision point is taken. Each activity must have a start and an end. Once again, consistency analysis is enough for TPNs as they do not deal with any uncontrollable part.

Moreover, STNs, Labeled STNs and TPNs don't specify any uncontrollable part (consistency analysis is enough). Therefore, it is not really possible to compare them with CSTNUDs.

Temporal Plan Networks with Uncertainty (*TPNUs*, [20]) extend TPNs by addressing both controllable and uncontrollable choices. However, TPNUs still suffer from a structured approach and they do not deal with uncontrollable durations. CSTNUDs don't have to follow a structured approach and they also deal with uncontrollable durations simultaneously.

Several different extensions of STNs have been proposed to address different kinds of uncertainty. Among those, we discuss here CSTNs, STNUs and CSTNUs.

A *CSTN* [3] extends an STN by addressing uncontrollable choices only. The dynamic controllability problem for CSTN is in PSPACE [21].

An *STNU* [6] extends an STN by adding contingent links only modeling uncontrollable (but bounded) durations. The dynamic controllability problem for STNUs is in PTIME [5].

A *CSTNU* [8] merges the semantics of STNUs and CSTNs to deal with both conditional constraints and uncontrollable durations simultaneously. In [7], Combi et al. proposed an algorithm for checking the dynamic controllability of a CSTNU extending the algorithm for STNUs given in [5]. The algorithm is sound but not complete and the complexity of the dynamic controllability problem is still unknown.

CSTNs, STNUs and CSTNUs don't employ decisions, therefore they are unable to influence any uncontrollable part they specify.

There exist two main classes of algorithms for checking the dynamic controllability of a temporal network subject to uncertain durations and conditionals: network-based algorithms and TGA-based algorithms.

Network-based algorithms take as input such temporal networks in their labeled distance graph representation and propagate constraints by analyzing triangles of edges (i.e., node triples) generating new constraints (i.e., adding

new edges) depending on a few finite preconditions. If the algorithm terminates before a cutoff bound is reached (or a negative cycle is detected), then the network is dynamically controllable.

TGAs allow one to work with sound and complete DC-checking algorithms. According to [3] the definition of soundness for a DC CSTNU is: "If the algorithm says no, then the network is not DC", whereas that of completeness is "for each non DC network, the algorithm says no" (equivalently, "if the algorithm says yes, then the network is DC"). The DC-checking of a CSTNU is implemented by first encoding a given CSTNU into an equivalent TGA, where the environment is assigned to controllable transitions and the controller to uncontrollable ones. Then, a control strategy for the environment—via Timed Computation Tree Logic (TCTL, [22]) model checking—is searched to prevent the controller from entering a special location of interest. If the environment succeeds, then the network is not dynamically controllable, otherwise the network is DC. The correctness of the approach for CSTNUs (which embed all previous formalisms) is proved in [15]. We built CSTNUDs on top of CSTNUs to inherit the same sound and complete approach.

In [9], CSTNs are extended with decision nodes regulating the truth value assignment for some propositions under control. That work focuses on the complexity analysis of the DC-checking problem proving that it is PSPACE-complete and provides algorithms for special cases in which either the network specifies only decisions and no observations or all decisions are made *before* any observation. In this paper we followed a completely different direction starting from CSTNUs (thus also considering contingent links) and it is based on TGAs.

In [23], STNUs are extended with security constraints in order to model temporal role-based access controlled workflows in which authorization constraints and temporal constraints mutually influence one another. Controllability checking has not been addressed for such an extension.

Access Controlled Temporal Networks (*ACTNs*, [18]) extend CSTNUs to represent a dynamic user assignment that also depends on temporal aspects. ACTNs employ disjunctive constraints. CSTNUDs do not employ disjunctive constraints (for the same scenario) nor allow for resource scheduling.

In temporal workflow management, the difference between controllable and uncontrollable XOR splits is introduced in [24] and a technique based on PERT-nets computes internal activity deadlines in order to meet the global ones. Some missed deadlines require human interaction for recovery. We rely

on DC, which guarantees that we never miss any deadline.

In [25], UPPAAL-TIGA is used to synthesize a controller for timeline-based plans that consider multivalued state variables and networks of TGAs. Apart from time points, our variables are Boolean and our encoding involves one TGA only.

Weak, strong and dynamic controllability are investigated for access-controlled workflows under conditional uncertainty in [26]. That work deals with structured workflows by unfolding workflow paths, considering binary constraints only (whose labels are always the conjunction of the labels of the connected tasks) and assuming that a total order for the tasks is given in input.

Constraint networks under conditional uncertainty (*CNCUs*, [27]) have been recently provided as a new interesting line of research for a dynamic resource scheduling under uncertainty. CNCUs deal with conditional uncertainty only and have been strongly inspired from CSTNs. CSTNUDs do not deal with resource scheduling.

10. Conclusions and Future Work

We introduced Conditional Simple Temporal Networks with Uncertainty and Decisions (CSTNUDs). We modeled the DC-checking of a CSTNUD as a two-player game, where `Player1` models the controller and `Player2` models the environment and we gave the execution semantics in move-based strategies. We provided an encoding from CSTNUDs into TGAs as an optimized extension of that given for CSTNUs and discussed the correctness and complexity of such an encoding.

We discussed the expressiveness of CSTNUDs by providing a hierarchy of simple temporal networks. We proved that CSTNUDs can embed all minor temporal network formalisms such as STNs (if $\mathcal{L} = \mathcal{OT} = \mathcal{DT} = \emptyset$), CSTNs (if $\mathcal{L} = \mathcal{DT} = \emptyset$), STNUs (if $\mathcal{OT} = \mathcal{DT} = \emptyset$), CSTNUs (if $\mathcal{DT} = \emptyset$), STNDs (if $\mathcal{L} = \mathcal{OT} = \emptyset$), CSTNDs (if $\mathcal{L} = \emptyset$), and STNUDs (if $\mathcal{OT} = \emptyset$).

Finally, we developed Esse, a tool for CSTNUDs, to automate the approach and used it to validate a temporal workflow for a goods delivery process translated into a corresponding CSTNUD. If a CSTNUD is DC, Esse saves to file a memoryless execution strategy to later carry out an arbitrary number of execution simulations (validate once, execute anytime). We also compared the experimental phase done in the paper on which this work is based and found that Esse performs better overall.

As future work, we plan to address weak and strong controllability of CSTNUDs (along with their related complexities), and develop dedicated algorithms (e.g., propagation-based) for CSTNUDs and for the new formalisms STNDs, CSTNDs and STNUDs.

References

[1] R. Dechter, I. Meiri, J. Pearl, Temporal constraint networks, Artificial Intelligence 49 (1-3) (1991) 61–95.

[2] P. R. Conrad, B. C. Williams, Drake: An efficient executive for temporal plans with choice, Journal of Artificial Intelligence Research 42 (1) (2011) 607–659.

[3] L. Hunsberger, R. Posenato, C. Combi, A sound-and-complete propagation-based algorithm for checking the dynamic consistency of conditional simple temporal networks, in: 22nd International Symposium on Temporal Representation and Reasoning (TIME 2015), IEEE, 2015, pp. 4–18. doi:10.1109/TIME.2015.26.

[4] I. Tsamardinos, T. Vidal, M. E. Pollack, CTP: A new constraint-based formalism for conditional, temporal planning, Constraints 8 (4) (2003) 365–388.

[5] P. Morris, N. Muscettola, Temporal Dynamic Controllability Revisited, in: 20th National Conference on Artificial Intelligence and the 17th Innovative Applications of Artificial Intelligence Conference (AAAI 2005), AAAI Press, 2005, pp. 1193–1198.

[6] P. H. Morris, N. Muscettola, T. Vidal, Dynamic control of plans with temporal uncertainty, in: 17th International Joint Conference on Artificial Intelligence (IJCAI 2001), Morgan Kaufmann, 2001, pp. 494–499.

[7] C. Combi, L. Hunsberger, R. Posenato, An algorithm for checking the dynamic controllability of a conditional simple temporal network with uncertainty, in: 5th International Conference on Agents and Artificial Intelligence (ICAART 2013), ScitePress, 2013, pp. 144–156. doi:10.5220/0004256101440156.

[8] L. Hunsberger, R. Posenato, C. Combi, The Dynamic Controllability of Conditional STNs with Uncertainty, in: Workshop on Planning and Plan Execution for Real-World Systems (PlanEx) at ICAPS 2012. http://arxiv.org/abs/1212.2005, 2012.

[9] M. Cairo, C. Combi, C. Comin, L. Hunsberger, R. Posenato, R. Rizzi, M. Zavatteri, Incorporating decision nodes into conditional simple temporal networks, in: 24th International Symposium on Temporal Representation and Reasoning (TIME 2017), LIPIcs, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.TIME.2017.9.

[10] M. Zavatteri, Conditional simple temporal networks with uncertainty and decisions, in: 24th International Symposium on Temporal Representation and Reasoning (TIME 2017), LIPIcs, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.TIME.2017.23.

[11] O. Maler, A. Pnueli, J. Sifakis, On the synthesis of discrete controllers for timed systems, in: 12th Annual Symposium on Theoretical Aspects of Computer Science (STACS 1995), Springer, 1995, pp. 229–242.

[12] A. Cimatti, L. Hunsberger, A. Micheli, R. Posenato, M. Roveri, Sound and complete algorithms for checking the dynamic controllability of temporal networks with uncertainty, disjunction and observation, in: 21st International Symposium on Temporal Representation and Reasoning (TIME 2014), IEEE, 2014, pp. 27–36. doi:10.1109/TIME.2014.21.

[13] R. Alur, D. L. Dill, A theory of timed automata, Theoretical Computer Science 126 (2) (1994) 183–235.

[14] J. E. Hopcroft, J. D. Ullman, Introduction to Automata Theory, Languages and Computation, Addison-Wesley, 1979.

[15] A. Cimatti, L. Hunsberger, A. Micheli, R. Posenato, M. Roveri, Dynamic controllability via timed game automata, Acta Informatica 53 (6-8) (2016) 681–722. doi:10.1007/s00236-016-0257-2.

[16] A. Cimatti, L. Hunsberger, A. Micheli, M. Roveri, Using timed game automata to synthesize execution strategies for simple temporal networks with uncertainty, in: 28th Conference on Artificial Intelligence (AAAI 2014), AAAI Press, 2014, pp. 2242–2249.

[17] P. Morris, The mathematics of dispatchability revisited, in: 26th International Conference on International Conference on Automated Planning and Scheduling (ICAPS 2016), AAAI Press, 2016, pp. 244–252.

[18] C. Combi, R. Posenato, L. Viganò, M. Zavatteri, Access controlled temporal networks, in: 9th International Conference on Agents and Artificial Intelligence (ICAART 2017), ScitePress, 2017, pp. 118–131. doi:10.5220/0006185701180131.

[19] P. Kim, B. C. Williams, M. Abramson, Executing Reactive, Model-based Programs through Graph-based Temporal Planning, in: 17th International Joint Conference on Artificial Intelligence (IJCAI 2001), Morgan Kaufmann, 2001, pp. 487–493.

[20] S. J. Levine, B. C. Williams, Concurrent Plan Recognition and Execution for Human-Robot Teams, in: 24th International Conference on Automated Planning and Scheduling (ICAPS 2014), AAAI Press, 2014, pp. 490–498.

[21] M. Cairo, R. Rizzi, Dynamic controllability of conditional simple temporal networks is pspace-complete, in: 23rd International Symposium on Temporal Representation and Reasoning (TIME 2016), IEEE, 2016, pp. 90–99. doi:10.1109/TIME.2016.17.

[22] R. Alur, C. Courcoubetis, D. Dill, Model-checking in dense real-time, Information and Computation 104 (1) (1993) 2–34.

[23] C. Combi, L. Viganò, M. Zavatteri, Security constraints in temporal role-based access-controlled workflows, in: 6th Conference on Data and Application Security and Privacy, (CODASPY 2016). http://doi.acm.org/10.1145/2857705.2857716, ACM, 2016, pp. 207–218.

[24] J. Eder, E. Panagos, H. Pozewaunig, M. Rabinovich, Time Management in Workflow Systems, Springer, 1999, pp. 265–280.

[25] A. Orlandini, A. Finzi, A. Cesta, S. Fratini, TGA-Based Controllers for Flexible Plan Execution, in: Advances in Artificial Intelligence: 34th Annual German Conference on AI (KI 2011), Springer, 2011, pp. 233–245.

[26] M. Zavatteri, C. Combi, R. Posenato, L. Viganò, Weak, strong and dynamic controllability of access-controlled workflows under conditional uncertainty, in: 15th International Conference on Business Process Management (BPM 2017), Springer, 2017, pp. 235–251. doi:10.1007/978-3-319-65000-5_14.

[27] M. Zavatteri, L. Viganò, Constraint networks under conditional uncertainty, in: 10th International Conference on Agents and Artificial Intelligence (ICAART 2018), SciTePress, 2018, pp. 41–52. doi:10.5220/0006553400410052.